Guide Pratique EDI NetBeans

Copyright © 2005 Sun Microsystems, Inc. All rights reserved.

Débogage Applications Java

Table of Contents

Starting a Debugging Session....2
Debugger Windows....3
Attaching the Debugger to a Running Application....5
Starting the Debugger Outside of the Project's Main Class....7
Stepping Through Code....7
Executing Code Line By Line....7
Executing a Method Without Stepping Into It....7
Resuming Execution Through the End of a Method....8
Continuing to the Next Breakpoint....8
Continuing to the Cursor Position....8
Stepping Into the JDK and Other Libraries....8
Limiting the Classes That You Can Step Into For a Library....8
Setting Breakpoints....9
Setting a Line Breakpoint....10
Setting a Breakpoint on a Class Call....10
Setting a Breakpoint on a Method or Constructor Call....11
Setting a Breakpoint on an Exception....11
Setting a Breakpoint on a Field or Local Variable....11
Setting a Breakpoint on the Start or Death of a Thread....12
Managing Breakpoints....12
Grouping Related Breakpoints....12
Enabling and Disabling Breakpoints....13
Deleting a Breakpoint....13
Customizing Breakpoint Behavior....13
Logging Breakpoints Without Suspending Execution....13
Customizing Console Messages When Breakpoints Are Hit....14
Making Breakpoints Conditional....14
Working With Variables and Expressions....15
Setting a Watch on a Variable or Field....16
Monitoring the Object Assigned to a Variable....16
Displaying the Value of a Class's toString Method....16
Changing Values of Variables or Expressions....17
Displaying Variables From Previous Method Calls....17
Backing up From a Method to its Call....17
Monitoring and Controlling Execution of Threads....18
Switching the Currently Monitored Thread....18
Suspending and Resuming Threads....18
Suspending a Single Thread at a Breakpoint....19
Isolating Debugging to a Single Thread....19
Fixing Code During a Debugging Session....19
Viewing Multiple Debugger Windows Simultaneously....20

MCours.com

L'EDI NetBeans fournit un environnement riche pour le règlement de problèmes et l'optimisation de vos applications. Le support de débogage intégré vous permet d'avancer pas à pas dans votre code et de surveiller les aspects de l'application, comme les valeurs de variables, la séquence actuelle d'appels de méthodes, le status des différents threads, et la création d'objets.

Lorsque vous utilisez le débogueur de l'EDI, il n'y a aucun raison pour vous de parsemer votre code avec des instructions `System.out.println` pour diagnostiquer tous problèmes survenant dans votre application. Au lieu de cela, vous pouvez utiliser le débogueur pour désigner les points d'intérêt dans votre code avec des points d'arrêts (qui sont stoqués dans l'EDI, pas dans votre code), suspend votre programme à ces points d'arrêt, et utilise les nombreuses fenêtres de débogage pour évaluer l'état du programme qui s'exécute.

De plus, vous pouvez modifier le code tout en le déboguant et recharger dynamiquement la classe dans le débogueur sans avoir à redémarrer la session de débogage.

Ce qui suit sont quelques unes des choses que vous pouvez faire dans le débogueur de l'EDI:

- Avancer ligne par ligne dans l'application.
- Avancer pas à pas dans le code source du JDK.
- Exécuter des morceaux de code bien spécifiques à un moment donné (en utilisant les points d'arrêt comme délimiteurs).
- Suspendre l'exécution lorsqu'une condition que vous avez spécifiée est rencontrée (comme par exemple quand un itérateur atteint une certaine valeur).
- Suspendre l'exécution lors d'une exception, soit à la ligne de code qui a provoqué l'exception, ou dans l'exception elle-même.
- Pister la valeur d'une variable ou d'une expression.
- Pister l'objet référencé par une variable (fixed watch).
- Fixer le code à la volée et continuer la session de débogage.
- Suspendre les threads individuellement ou collectivement.
- Revenir au début d'une méthode appelée précédemment (pop a call) dans la call stack actuelle.
- Exécuter de multiple sessions de débogage en même temps. Par exemple, vous pourriez recourrir à cette capacité pour déboguer une application client/serveur.

Démarrer une Session de Débogage

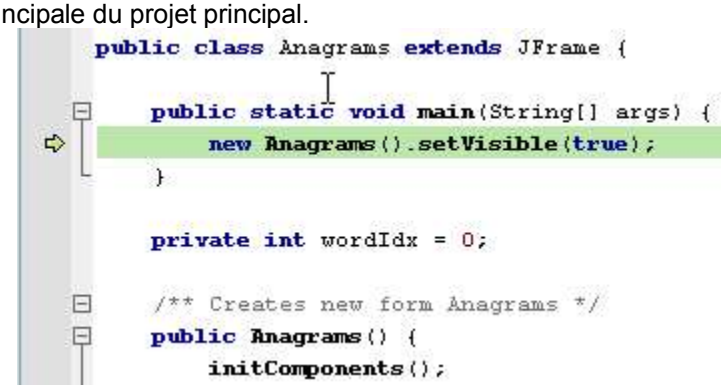
Le moyen le plus simple de lancer le débogueur est de choisir Run | Step Into. Le marqueur du programme (indiqué par un fond vert fluorescent et l'icone) s'arrête sur la première ligne de la méthode main de votre projet principal.

Vous pouvez alors avancer pas à pas dans votre programme, de façon incrémentielle, avec n'importe laquelle des commandes Step pour observer le flux du programme et surveiller les différentes valeurs des variables dans la fenêtre Variables Locales. Voyez le Tableau 4 pour une description de toutes les commandes Step et les sujets qui s'ensuivent pour savoir comment tirer profit des capacités du débogueur.

Conseil EDI NetBeans

Vous pouvez également utiliser la commande Run To Cursor pour démarrer une session de débogage. Dans l'Editeur de Source, cliquez

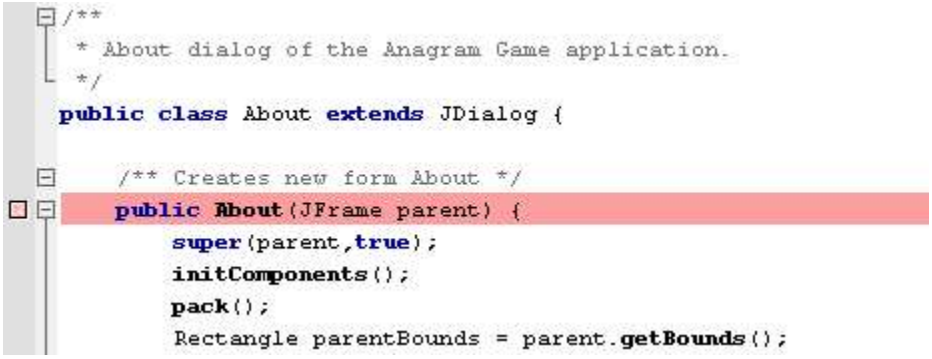
sur une ligne où vous désirez suspendre l'exécution et choisissez Run | Run To Cursor. Cette commande ne fonctionne, pour lancer une session de débogage, que si vous sélectionnez une ligne de code dans la class principal du projet principal ou une classe appelée directement depuis la classe principale du projet principal.



Plus que probablement, vous désirerez avancer pas à pas dans le code à partir d'un certain point. Dans ce cas, vous pouvez spécifier certains points d'arrêts dans le programme où il faudra suspendre l'exécution et ensuite lancer le débogueur. Pour faire cela, il vous faut :

1. Mettre un point d'arrêt dans votre projet principal en ouvrant la classe dans l'Editeur de Source et en cliquant dans la marge de gauche à hauteur de la ligne ou vous désirez mettre un point d'arrêt (ou presser Ctrl-F8)..

   Vous savez que le point d'arrêt est bien défini par l'apparition du glyphe rose ◻ dans la marge et le fait que la ligne apparait avec un fond rose.



2. Pressez F5 pour démarrer le débogage du projet principal.

Lorsque l'exécution du programme s'arrête au point d'arrêt (ce que vous constaterez par le fait que le point d'arrêt rose est remplacé par le marqueur vert), vous pouvez avancer pas à pas dans le code tout en visualisant le contenu des variables, threads et d'autres informations..

Voyez les sujets s'y relatant pour plus de détails sur le pas à pas et visualiser les informations du programme..

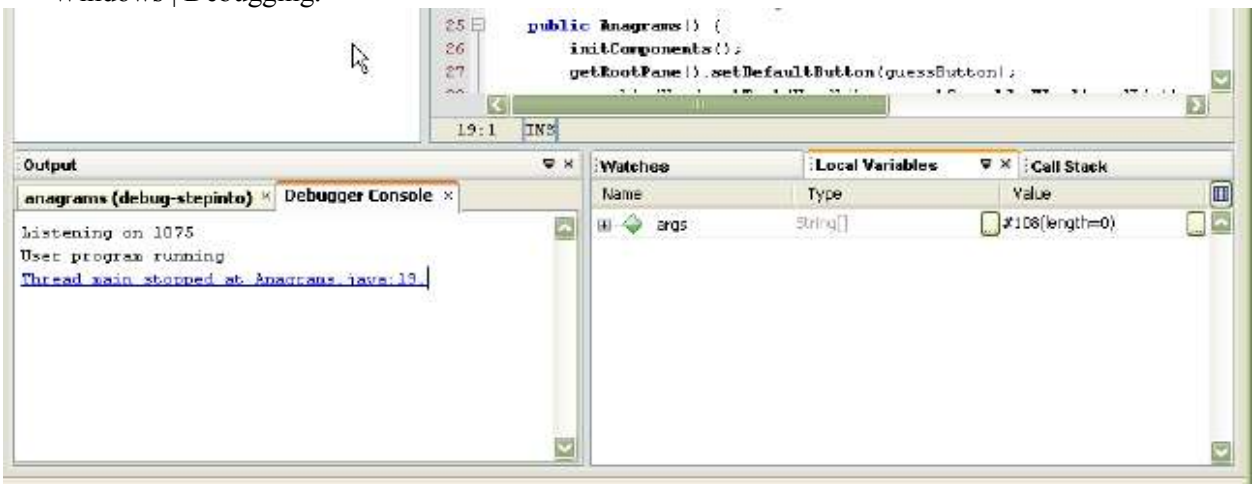
Conseil EDI NetBeans

Si vous avez créé un projet libre, vous devez créer des liens entres les tâches Ant et la commande Debug Project pour pouvoir déboguer. Voir le chapitre Intégration de Scripts Ant Existants Avec l'EDI pour plus de détails.

Fenêtre Débogage

Lorsque vous déboguer un programme, la console Debogage apparaît dans un onglet dans la partie inférieure gauche de l'EDI. La console de Débogage enregiste le statut d'exécution du programme débogué (comme par exemple si le code est arrêté à un point d'arrêt). De plus, un onglet s'ouvre dans la fenêtre Output pour y enregistrer tout output de l'application (ainsi que le résultat du script Ant que l'EDI utilise pour exécuter la commande).

Dans le coin inférieur droit, certaines fenêtres (Watches, Local Variables, et Call Stack) s'ouvrent en tant qu'onglet et fournissent certaines informations concernant la session de débogage, comme la valeur actuelle des variables, et la liste des méthodes appelées. Vous pouvez également ouvrir les fénêtres de débogage individuellement en les choisissant depuis le menu Windows | Debugging.



La plupart des fenêtres affichent des informations selon le *contexte* actuel du débogueur. En général, le context actuel correspond à un appel de méthode dans un thread dans une session. Vous pouvez changer le contexte (par exemple, désigner un autre thread dans la fenêtre Thread) sans affecter la façon dont le programme débogué s'exécute.

Voyez le Tableau 1 pour la liste de toutes les fenêtres disponibles et comment les ouvrir.

Tableau 1: Fenêtres Débogueur

Fenêtre Débogueur	**S'ouvre Avec**	**Description**
Local Variables (Variables Locales)	Maj-Alt-1 (or Window | Debugging | Local Variables)	Affiche tous les champs et variables locales dans le contexte actuel du développeur et leurs valeurs actuelles. Les champs sont repris sous le noeud `this`.
Watches (Surveillance)	Maj-Alt-2 (or Window | Debugging | Watches)	Affiche les noms des champs, variables locales, ou expressions que vous avez mis sous surveillance. Bien que toutes vos alarmes sont affichées quelque soit le contexte actuel, la valeur affichée est la valeur dans le contexte actuel (par pour le contexte dans lequel l'alarme fut défini). Par exemple, si vous avez une alarme sur le mot-clef `this`, ce `this` référencé dans la fenêtre Watches va toujours correspondre à l'objet référencé dans l'appel de méthode actuelle.
Call Stack (Pile d'appel)	Maj-Alt-3 (or Window | Debugging | Call Stack)	Affiche tous les appels de méthodes dans la chaine courante d'appels. La fenêtre de pile d'appel vous permet d'afficher directement le code d'une méthode sélectionnée, de refaire l'exécution du programe depuis l'appel de méthode précédent, ou de sélectionner un contexte pour voir les valeurs des variables locales.

MCours.com

Fenêtre Débogueur	S'ouvre Avec	Description
Classes (Classes)	Maj-Alt-4 (or Window | Debugging | Classes)	Fournit une vue arborescente de classes pour l'application actuellement déboguée, regroupée par classloader.
Breakpoints (points d'arrêt)	Maj-Alt-5 (or Window | Debugging | Breakpoints)	Affiche tous les points d'arrêts définis dans toutes les sessions de débogage en cours d'exécution.
Threads	Maj-Alt-6 (or Window | Debugging | Threads)	Affiche tousles threads de la session actuelle. Dans cette fenêtre, vous pouvez changer de contexte en désignant un autre thread comme thread actuel.
Sessions	Maj-Alt-7 (or Window | Debugging | Sessions)	Affiche un noeud pour chaque session de débogage en cours. Depuis cette fenêtre, vous pouvez changer de session actuelle.
Sources	Maj-Alt-8 (or Window | Debugging | Sources)	Affiche les sources qui sont disponibles pour le débogage et vous permet de spécifier lesquels utiliser. Par exemple, vous pouvez utiliser cette fenêtre pour permettre le débogage avec les sources du JDK.

Attacher le Débogueur à une Application en cours d'Exécution

Si vous devez déboguer une application qui s'exécute sur une autre machine ou qui s'exécute dans une machine virtuelle différente, vous pouvez attacher le débogueur de l'EDI à cette application:

1. Démarrer l'application que vous désirez déboguer en mode debug. Cela se fait en rajoutant certains arguments spécifiques au script qui lance l'application.

   Pour les utilisateurs de Windows qui utilisent un JDK de Sun, la liste d'arguments peut ressembler à ceci (tout sur une seule ligne, et pas d'espace après `-Xrunjdwp:`):

   ```
   java -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:
   transport=dt_shmem,server=y,address=MyAppName,suspend=n
   -classpath C:\my_apps\classes mypackage.MyApp
   ```

   Sous d'autres systèmes d'exploitation, la liste d'argument peut ressembler à ceci:

   ```
   java -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:
   transport=dt_socket,server=y,address=8888,suspend=n
   -classpath HOME/my_apps/classes mypackage.MyApp
   ```

   Pour une documentation plus complete de ces options, visitez la page http://java.sun.com/products/jpda/doc/conninv.html.
2. Dans l'EDI, ouvrez le projet qui contient le code source de l'application à déboguer.
3. Sélectionnez Run | Attach Debugger.
4. Dans la boite de dialogue Attach, sélectionnez le connecteur depuis la liste déroulante Connector.

   Sélectionnez SharedMemoryAttach si vous désirez attacher une application qui a été lancée avec le transport `dt_shem`. Sélectionnez SocketAttach si vous désirez attacher une application qui a été lancée avec le transport `dt_socket`.

Voyez la ligne Connector du Tableau 3 pour plus d'informations sur les différents types de connecteurs.

5. Remplisser le restant des champs. Les champs qui apparaissent après Connector dépend du type de connecteur que vous avez sélectionné. Voyez le Tableau 2 pour les differents champs.

Tableau 2: Paramètres de Lancement du Débogueur

Paramètres de Lancement ou Sous-Paramètres	***Description***
-Xdebug	Permet à l'application d'être déboguée.
-Xnoagent	Désactive le `sun.tools.debug` pour que le débogueur JPDA puisse attacher son propre agent.
-Djava.compiler	Désactive le compilateur JIT (Just-In-Time).
-Xrunjdwp	Charger l'implémentation de référence de Java Debug Wire Protocol, qui permet le débogage à distance.
transport	Nom du transport à utiliser pour déboguer l'application. La valeur peut être `dt_shmem` (pour une connection mémoire partagée) ou `dt_socket` (pour une connection socket). Les connections mémoires partagées ne sont possibles que sur des machines Windows.
server	Si cette valeur vaut n, l'application essaye de s'attacher au débogueur à l'adresse spécifiée dans le sous-paramètre `address`. Si cette valeur vaut `y`, l'application écoute pour la connection à cette adresse.
address	Pour les connections par socket, spécifie un numéro de port pour la communication entre le débogueur et l'application. Pour les connections par mémoire partagée, spécifie un nom qui se réfère à la mémoire partagée à utiliser. Ce nom peut consister en toute combinaison de caractères qui sont valides dans des noms de fichiers sur une machine Windows, à l'exception de la backslash (\). Vous utiliserez alors ce nom dans le champ Name de la boite de dialogue Attach lorsque vous attacherez le débogueur à l'application.
suspend	Si la valeur est `n`, l'application démarre immédiatement. Si la valeur est `y`, l'application attend jusqu'à ce qu'un débogueur soit attaché avant de s'exécuter.

Tableau 3: Champs Boite de Dialogue Attach

Champ	**Description**
Connector	Spécifie le type de connecteur JPDA à utiliser. Sur des machines Windows, vous pouvez choisir entre les connecteurs mémoire partagée et les connecteurs par socket. Sur d'autres systèmes, vous ne pouvez utiliser qu'un connecteur par socket. For both shared memory connectors and socket connectors, there are Attach and Listen variants. You can use an Attach connector to attach to a running application. Vous pouvez utiliser un connecteur Listen si vous désirez que l'application qui s'exécute pour initier la connection au débogueur. Si vous utilisez le connecteur Listen, plusieurs pplications tournant sous différentes JVMs peuvent se connecter au débogueur.
Transport	Spécifier le protocole de transport JPDA à utiliser. Ce champ est automatiquement remplis selon ce que vous avez remplit dans le champ Connector.
Host	(Uniquement pour les connections par socket de type attach.) Le host name de l'ordinateur où l'application déboguée s'exécute.

Champ	Description
Port	(Uniquement puor les connections par socket.) Le numéro de port auquel l'application s'attache ou écoute. Vous pouvez assigner un numéro de port dans le sous-paramètre `address` du paramètre `Xrunjdwp` qe vous passez à la JVM de l'application à déboguer. Si vous n'utilisez pas cette sous-option, un numéro de port est assigné automatiquement, et vous pouvez déterminer le numéro de port assigné en regardant à la console d'output du processus.
Timeout	Le nombre de millisecondes que le débogueur attend pour établir la connection.
Name	(Uniquement pour les connections par mémoire partagée.) Spécifie la mémoire partagée à utiliser pour la session de débogage. Cette valeur doit correspondre à la valeur du sous-paramètre `address` du paramètre `Xrunjdwp` que vous avez passé à la JVM de l'application qui doit être déboguée.
Local Address	(Uniquement pour la connection par Socket de type Listen.) Le host name de l'ordinateur sur lequel vous tournez.

Lancer le Débogueur En Dehors de la Classe Principale du Projet

Si vous avez plusieurs classes exécutables dans votre projet, il y a des fois où vous aimeriez lancer le débogueur pour une classe différente de celle qui est spécifiée comme la classe principale du projet.

Pour lancer le débogueur sur une classe autre que la classe principale du projet, cliquez-droit sur le noeud du fichier dans la fenêtre Projects ou Files et choisissez Debug File.

Vous ne pouvez démarrer le débogueur sur un fichier que s'il possede une méthode `main`.

Avancer pas à pas dans le code

Une fois que l'exécution de votre programme est suspendu, vous avez plusieurs façons de continuer l'exécution du code. Vous pouvez avancer ligne par ligne (Step In) ou avec un incrément plus élevé. Voyez le tableau 4 pour les commandes disponibles pour l'exécution pas à pas ou continue .

Tableau 4: Commandes Pas à Pas

Commande	Description
Step Into (F7) (Ligne suivante)	Exécute la ligne courante. Si la ligne est l'appel d'une méthode ou d'un constructeur et que les sources soient disponibles pour le code appelé, le marqueur du programme va se positionner à la déclaration de la méthode ou du constructeur. Autrement, le marqueur se positionnera à la ligne suivante du fichier.
Step Over (F8)	Exécute la ligne courante et déplacer le marqueur du programme à la ligne suivante du fichier. Si la ligne exécutée est l'appel d'une méthode ou d'un constructeur, le code dans la méthode ou le constructeur est également exécuté.
Step Out (Alt-Shift-F7)	Exécute le restant du code dans la méthode ou constructeur courant et déplace le marqueur du programme à la ligne après l'appel de la méthode ou du constructeur. Cette commande est utile si vous êtes à l'intérieur d'une méthode que vous ne devez pas analyser.
Run to Cursor (F4)	Exécute toutes les lignes du programme entre la ligne en cours et le curseur dans l'Editeur de Source..

Commande	Description
Pause	Arrête tous les threads de la session en cours.
Continue (Ctrl-F5)	Continue l'exécution du programme jusqu'au prochain point d'arrêt.

Exécution Code Ligne par Ligne

Vous pouvez déboguer ligne par ligne en choisissant Run | Step Into (F7). Si vous utilisez la commande Step Into sur un appel de méthode, le débogueur entre dans la méthode et s'arrête à la première ligne, à moins que la méthode fasse partie d'une bibliothèque que vous n'avez pas spécifié comme devant être utilisée par le débogueur. Voyez le sujet Exécution Dans le JDK et Autres Bibliothèques ci-dessous pour les détails quant à la façon de maquer les sources disponibles pour l'utilisation dans le débogueur.

Exécution d'une Méthode Sans Y Rentrer

Vous pouvez exécuter une méthode sans que le déboguer ne marque une pause dans la méthode, en choisissant run | Step Over (F8). Après avoir utilisé la commande Step Over, le débogueur marque à nouveau une pause après l'appel de méthode.

Terminer l'Exécution Jusqu'à la Fin d'une Méthode

Si vous êtes en train de déboguer une méthode que vous ne désirez plus analyser, vous pouvez demander au débogueur de continuer l'exécution de la méthode en d'ensuite s'arrêter que sur la ligne après l'appel de méthode.

Pour terminer l'exécution d'une méthode de cette façon, choisissez Run | Step Out Of (Maj-Alt-F7).

Continuer Jusqu'au Prochain Point d'Arrêt

Si vous ne devez pas observer chaque ligne de code lors du débogage, vous pouvez continuer l'exécution jusqu'au prochain point d'arrêt ou jusqu'à ce que l'exécution soit autrement suspendue.

Pour continuer l'exécution d'un programme qui a été suspendu à un point d'arrêt, choisissez choose Run | Continue ou pressez les touches Ctrl-F5.

Continuer Jusqu'à la Position du Curseur

Lorsque l'exécution est suspendue, vous pouvez continer jusqu'à une ligne spécifique sans devoir definir de point d'arrêt, en plaçant le curseur sur cette ligne et en choisissant Run | Run to Cursor (F4).

Exécution Dans le JDK et Autres Bibliothèques

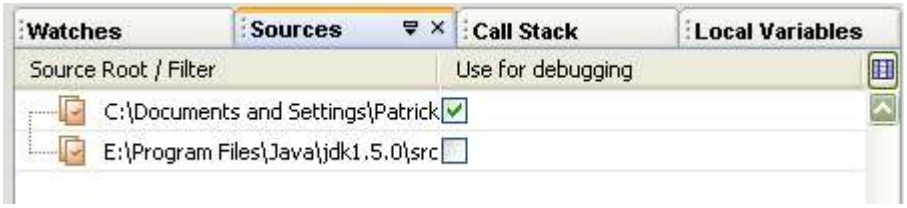
Lorsque vous êtes en train de déboguer, vous pouvez également déboguer le JDK et d'autres bibliothèques si vous en avez le code source associé. Voir le sujet Rendre les Sources Externes Disponibles dans l'EDI du chapitre Essentiel du Projet de l'EDI pour savoir comment associer le code source avec une bibliothèque.

Par défaut, l'EDI ne va pas déboguer les sources du JDK. Si vous utilisez la commande Step Into sur l'appel d'une méthode du JDK, l'EDI exécute la méthode et retourne le marqueur du

programme à la ligne suivant l'appel de méthode (comme si vous aviez utilisé la commande Step Over).

Pour activer le débogage dans les sources du JDK :

1. Lancer le débogueur pour l'application.
2. Ouvrez la fenêtre Sources en choisissant Window | Debugging | Sources ou en pressant les touches Maj-Alt-8.
3. Cocher la case Use For Debugging pour le JDK.



Limiter les Classes Auxquelles Vous Pouvez Accéder Pour une Bibliothèque

Si vous utilisez une bibliothèque pour le débogage, vous pouvez mettre un filtre pour exclure certaines sources.

Pour exclure des classes d'être utilisés dans le débogueur:

1. Démarrer le débogueur pour l'application.
2. Ouvrir la fenêtre Sources en choisissant Window | Debugging | Sources ou en pressant Maj-Alt-8.
3. Cliquez-droit sur la ligne de la bibliothèque pour laquelle vous désirez créer un filtre d'exclusion et choisissez Add Class Exclusion Filter.
4. Introduisez un filtre dans la boîte de dialogue Add Class Exclusion Filter.

Le filtre peut être :

- un nom de classe pleinement qualifié.
- Un nom de paquetage ou un nom de classe avec une astérisque (*) à la fin pour créer une wildcard. Par exemple, vous pourriez introduire ce qui suit pour exclure toutes les classes du paquetage `javax.swing`: `javax.swing.*`
- une expression avec une wildcard au début. Par exemple, pour exclure toutes les classes qui ont Test à la fin de leurs noms, vous pouvez utiliser: `*Test`

Vous pouvez créer plusieurs filtres d'exclusion de classes.

Pour désactiver le filtre, décocher la case Use in Debugging près du filtre dans la fenêtre Sources.

Pour supprimer un filtre d'exclusion de classes, cliquez-droit sur le filtre et choisissez Delete.

MCours.com

Définir des Points d'arrêts

Un point d'arrêt est un marqueur que vous définissez pour spécifier où l'exécution devrait être suspendue lorsque vous exécutez votre application dans le débogueur de l'EDI. Les points d'arrêt sont stoqués dans l'EDI (pas dans le code de votre application) et persistent entre les sessions de débogages et les sessions de l'EDI.

Lorsque l'exécution marque une pause sur un point d'arrêt, la ligne où l'exécution est suspendue est colorée en vert dans l'Editeur de Source et un message est affiché dans la Console de Débogueur avec l'information sur le point d'arrêt qui a été atteint.

Dans leur forme la plus simple, les points d'arrêt vous fournissent une façon de suspendre l'exécution du programme à un point spécifique pour que vous puissiez:

- Superviser les valeurs des variables à ce point dans l'exécution du programme.
- Prendre le contrôle de l'exécution du programme en avançant ligne par ligne dans le code, ou méthode par méthode.

Cependant, vous pouvez également utiliser les points d'arrêt comme outil de diagnostic pour faire des choses comme:

- Détecter lorsque la valeur d'un champ ou variable locale est modifiée (ce qui peut, par exemple, vous aider à déterminer quelle partie du code assigne une valeur inappropriée à un champ).
- Détecter lorsqu'un objet est créé (ce qui peut être utile, par exemple, lorsque vous essayer de pister un memory leak).

Vous pouvez définir plusieurs points d'arrêt et vous pouvez définir différents types de points d'arrêt. Le type le plus simple de point d'arrêt est un point d'arrêt de ligne, où l'exécution du programme s'arrête à la ligne spécifiée. Vous pouvez également définir des points d'arrêts pour d'autres situations, comme l'appel d'une méthode, le lancement d'une exception, ou le changement de valeur d'une variable. De plus, vous pouvez définir des conditions pour certains types de points d'arrêt pour qu'ils suspendent l'exécution du programme uniquement dans des circonstances bien spécifiques. Voyez le tableau 5 pour un résumé des types de points d'arrêt.

Tableau 5: Catégories de Points d'Arrêt

Type de Points d'Arrêt	**Description**
Line	Définit sur une ligne de code. Lorsque le débogueur atteint cette ligne, il s'arrête AVANT d'exécuter la ligne. Le point d'arrêt est marqué par une couleur d'arrière plan rose et l'icône . Vous pouvez également spécifier les conditions pour les points d'arrêt de ligne.
Class	L'Exécution est suspendu lorsque la classe est référencé depuis une autre classe et avant qu'une seule ligne de la classe ayant le point d'arrêt ne soit exécuté.
Exception	L'Exécution est supendu lorsqu'une exception se produit. Vous pouvez spécifier si l'exécution s'arrête sur des caught exceptions, uncaught exceptions, ou les deux.
Method	L'Exécution est suspendue lorsque la méthode est appelée.
Variable	L'Exécution est suspendue lorsque la variable est accédée. Vous pouvez également configurer le point d'arrêt pour que l'exécution ne soit suspendue que lorsque la variable est modifiée.
Thread	L'Exécution est suspendue lorsqu'un thread est démarré, terminé, ou les deux.

Définir un Point d'Arrêt sur une ligne

Pour définir un point d'arrêt de ligne, cliquez dans la marge gauche de la ligne où vous désirez mettre un point d'arrêt, ou positionnez le curseur dans la ligne et pressez Ctrl-F8.

Pour supprimer un point d'arrêt, cliquez à nouveau dans la marge de gauche de la ligne où se trouve le point d'arrêt, ou positionnez le curseur dans la ligne et pressez Ctrl-F8.

Si vous désirez personnaliser un point d'arrêt de ligne, vous pouvez le faire via la fenêtre Breakpoints. Choisissez Window | Debugging | Breakpoints ou pressez Maj-Alt-5. Dans la fenêtre Breakpoints, cliquez-droit sur le point d'arrêt et choisissez Customize.

Définir un Point d'Arrêt sur l'Appel d'une Classe

Vous pouvez définir un point d'arrêt sur une classe pour que le débogueur suspend l'exécution lorsque le code de la classe sera accédé et/ou lorsque la classe est enlevée de la mémoire.

Pour mettre un point d'arrêt sur une classe:

1. Choisissez Run | New Breakpoint (Maj-Ctrl-F8).
2. Dans la boîte de dialogue New Breakpoint, sélectionnez Class dans la liste déroulante Breakpoint Type.
3. Introduisez les noms de la classes et du paquetage. Ces champs devraient être automatiquement remplis avec la classe affichée actuellement dans l'Editeur de Source.

Conseil EDI NetBeans

Vous pouvez spécifier de nombreuses classes pour le point d'arrêt à appliquer, soit en utilisant les wildcards dans les champs Package Name et Class Name, ou en sélectionnant la case Exclusion Filter.

Utilisez l'astérisque (`*`) pour créer des wildcards dans les champs Package Name et Class Name si vous désirez que le point d'arrêt s'applique à plusieurs classes ou à toutes les classes dans un paquetage. Par exemple, si vous n'introduisez qu'une astérisque dans le champs Class Name, le point d'arrêt s'appliquera à toutes les classes du paquetage spécifié dans le champ Package Name. Vous pouvez utiliser l'astérisque au début ou à la fin de l'expression, mais pas au milieu.

Cochez la case Exclusion Filter si vous désirez que le point d'arrêt s'applique à toutes les classes (y compris les classes du JDK) excepté pour celles qui correspondent aux classes ou paquetages spécifiés dans les champs Package Name et Class Name. Vous pouvez définir plusieurs points d'arrêt avec le Exclusion Filter activé. Par exemple, vous pouvez définir un filtre d'exclusion sur `com.mydomain.mypackage.mylib.*` parce que vous désirez que le point d'arrêt de classe ne s'applique qu'à toutes vos classes excepté celles du paquetage `mylib`. Cependant, si vous ne désirez pas que le débogueur marque une pause au chargement de chaque classe du JDK appelée, vous devriez également mettre un point d'arrêt de classe avec un filtre d'exclusion sur `java.*`.

Définir un Point d'Arrêt sur l'Appel d'une Méthode ou d'un Constructeur

Vous pouvez définir un point d'arrêt pour que le débogueur suspende l'exécution du programme lorsqu'une méthode ou un constructeur est appelé avant-même qu'une ligne de code de la méthode ou du constructeur ne soit exécuté.

Pour définir un point d'arrêt sur une méthode ou un constructeur:

1. Choisissez Run | New Breakpoint (Maj-Ctrl-8).
2. Dans la boîte de dialogue New Breakpoint, sélectionnez Method dans la liste déroulante Breakpoint Type.
3. Introduisez les noms de classe, paquetage et méthode. Ces champs sont remplis automatiquement selon la classe ouverte dans l'Editeur de Source et la position du curseur.

Vous pouvez faire que le point d'arrêt s'applique à toutes les méthodes et constructeurs dans la classe en cochant la case All Methods For Given Classes.

Définir un Point d'Arrêt sur une Exception

Vous pouvez définir un point d'arrêt pour que le débogueur marque une pause lorsqu'un exception est lancée dans votre programme.

Pour définir un point d'arrêt sur une exception:

1. Choisissez Run | New Breakpoint (Maj-Ctrl-F8).
2. Dans la boite de dialogue New Breakpoint, sélectionnez Exception dans la liste déroulante Breakpoint Type.
3. Dans le champ Exception Class Name, sélectionnez le type d'exécution pour lequel vous aimeriez mettre un point d'arrêt.
4. Dans la liste déroulante Stop On, sélectionnez si vous désirez que le point d'arrêt s'applique lors d'un catch d'exception ou pas.

Définir un Point d'Arrêt sur un Champ ou une Variable Locale

Vous pouvez définir un point d'arrêt pour que le débogueur marque une pause lorsqu'un champ ou une variable est accédée (ou lorsque le champ ou la variable est modifiée).

Pour définir un point d'arrêt sur un champ ou une variable:

1. Choisissez Run | New Breakpoint (Maj-Ctrl-F8).
2. Dans la boite de dialogue New Breakpoint, sélectionnez Variable dans la liste déroulante Breakpoint Type.
3. Remplissez les champs Package Name, Class Name, et Field Name.
4. Sélectionnez une option dans la liste déroulante Stop On.

Si vous sélectionnez Variable Access, l'exécution est suspendue chaque fois que le champ ou la variable est accédée dans le code.

Si vous sélectionnez Variable Modification, l'exécution n'est suspendue que si le champ ou la variable est modifiée.

Conseil EDI NetBeans

La plupart des champs de la boite de dialogue New Breakpoint sont correctement remplis pour vous si vous avez sélectionné la variable avant de presser Maj-Ctrl-F8. Vous devez sélectionner le nom entier de

la variable pour que cela fonctionne. Autrement, vous devrez remplir les informations pour la méthode qui contient la variable.

Définir un Point d'Arrêt sur le Démarrage ou la Fin d'un Thread

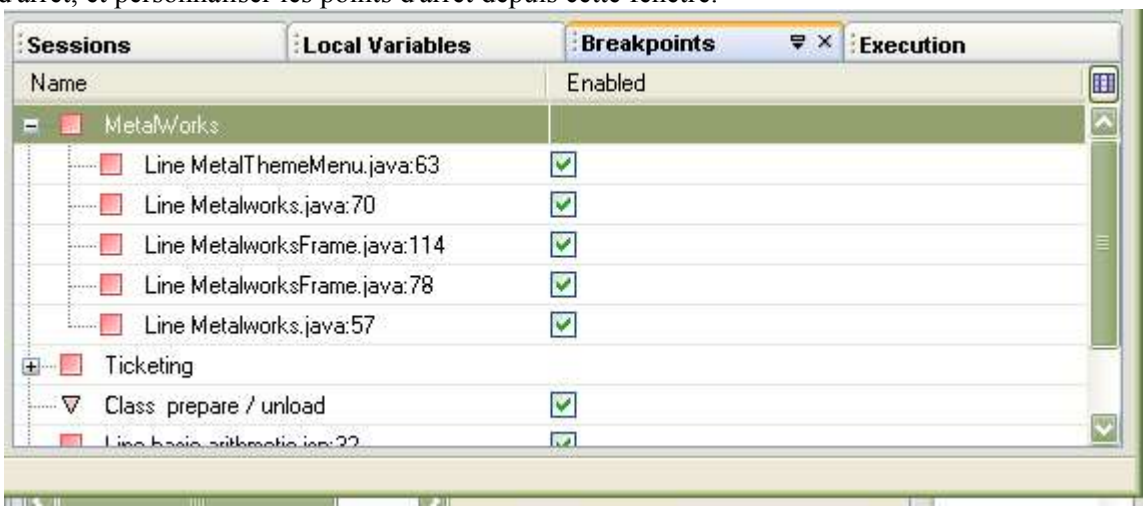
Vous pouvez superviser la création ou la fin des threads dans votre programme en créant un point d'arrêt qui aura son exécution suspendue à chaque fois qu'un nouveau thread est créé ou terminé.

Pour définir un point d'arrêt sur les threads:

1. Choisissez Run | New Breakpoint (Maj-Ctrl-F8).
2. Dans la boîte de dialogue New Breakpoint, sélectionnez Thread dans la liste déroulante Breakpoint Type.
3. Dans le champ Set Breakpoint On, sélectionnez Thread Start, Thread Death, ou Thread Start or Death.

Gestion des Points d'Arrêt

Vous pouvez utiliser la fenêtre Breakpoints (Maj-Alt-5) pour gérer les points d'arrêt depuis un seul endroit. Vous pouvez regrouper des points d'arrêt, désactiver temporairement des points d'arrêt, et personnaliser les points d'arrêt depuis cette fenêtre.



Grouper des Points d'Arrêt

Dans certains cas, vous pourriez avoir des points d'arrêts que vous aimeriez activer, désactiver ou supprimer en une fois. Ou peut-être que vous aimeriez rassembler certains points d'arrêt sous un même noeud pour rendre la fenêtre Breakpoints moins désordonnée.

Pour grouper certains points d'arrêt:

1. Ouvrez la fenêtre Breakpoints en choisissant Window | Debugging | Breakpoints (Maj-Alt-5).
2. Tout en maintenant la touche Maj ou Ctrl, cliquez sur les points d'arrêt que vous désirez grouper. Ensuite, cliquez-droit sur la sélection et choisissez Set Group Name.

Les points d'arrêt sont regroupés sous un noeud.

Activer et Désactiver des Points d'Arrêt

Vous pouvez trouver très utile de garder vos points d'arrêt, mais peut-être que vous ne désirez pas qu'ils soient tous actifs en même temps. Dans ce cas, vous pouvez désactiver un point d'arrêt ou un groupe de points d'arrêt tout en les préservant pour une utilisation ultérieure.

Pour désactiver un point d'arrêt ou un groupe de points d'arrêt:

1. Ouvrez la fenêtre Breakpoints en choisissant Window | Debugging | Breakpoints (ou en pressant Maj-Alt-5).
2. Dans la fenêtre Breakpoints, cliquez-droit sur le point d'arrêt ou le groupe de points d'arrêt et choississez disable.

Supprimer un Point d'Arrêt

Pour supprimer un point d'arrêt dans l'Editeur de Source, cliquez sur la marge de gauche de la ligne qui a le point d'arrêt ou cliquez dans la ligne et pressez Ctrl-F8.

Pour effacer un autre type de point d'arrêt:

1. Ouvrez la fenêtre Breakpoints en choissant Window | Debugging | Breakpoints (ou en pressant les touches Maj-Alt-5).
2. Dans la fenêtre Breakpoints, cliquez-droit sur le point d'arrêt et choisissez Delete.

Personaliser le Comportement du Point d'Arrêt

Il y a tout un tas de choses que vous pouvez personnaliser lorsqu'un point d'arrêt est atteint dans l'EDI et sur ce qui doit se passer dans l'EDI lorsque le point d'arrêt est atteint. Les chapitres suivants en couvrent quelques unes.

Logging Point d'Arrêt Sans Suspendre l'Exécution

Si vous désirez surveiller lorsqu'un point d'arrêt est atteint sans suspendre l'exécution à chaque fois que le point d'arrêt est atteint, vous pouvez configurer le point d'arrêt pour qu'il ne provoque pas la suspension de l'exécution. Lorsqu'un tel point d'arrêt est atteint dans le code, un message est affiché dans la Console Débogueur.

Pour empêcher la suspension de l'exécution lorsqu'un point d'arrêt est atteint:

1. Ouvrez la fenêtre Breakpoints en choisissant Window | Debugging | Breakpoints (Maj-Alt-5).
2. Dans la fenêtre Breakpoints, double-cliquez sur le point d'arrêt pour ouvrir la fenêtre Customize Breakpoint. (Dans l'Editeur de Source, cliquez-droit sur le point d'arrêt et sélectionnez Customize.)
3. Dans la liste déroulante Action, choisissez No Thread (Continue).

Personnaliser les Messages vers la Console Lorsqu'un Point d'Arrêt est Atteint

Vous pouvez personnaliser le texte qui est affiché à la console lorsqu'un point d'arrêt est atteint dans votre code.

Pour personnaliser le message de la console qui est imprimé lorsqu'un point d'arrêt est atteint:

1. Ouvrez la fenêtre Breakpoints en choisissant Window | Debugging | Breakpoints (Maj-Alt-5).
2. Dans la fenêtre Breakpoints, double cliquez sur le point d'arrêt pour ouvrir la fenêtre Customize Breakpoint. (Dans l'Editeur de Source, cliquez-droit sur le point d'arrêt et choisissez Customize.)
3. Dans le champ Print Text, modifier le texte que vous désirez imprimer.

Pour rendre le texte plus significatif, vous pouvez utiliser du code de substitution pour avoir des informations comme le nom du thread et le numéro de la ligne.

Tableau 6: Code de Substitution Pour le Texte Console du Point d'Arrêt

Code de Substitution	**Affiche**
`{className}`	Le nom de la classe où se situe le point d'arrêt. Ce code ne fonctionne pas pour les points d'arrêt de thread.
`{lineNumber}`	Le numéro de ligne où se situe le point d'arrêt. Ce code ne fonctionne pas pour les points d'arrêt de thread.
`{methodName}`	La méthode où se situe le point d'arrêt. Ce code ne fonctionne pas pour les points d'arrêt de thread.
`{threadName}`	Le thread dans lequel le point d'arrêt est atteint.
`{variableValue}`	La valeur de la variable (pour les points d'arrêt définis sur des variables) ou la valeur de l'exception (pour les points d'arrêt définis sur les exceptions).
`{variableType}`	Le type de variable (pour les points d'arrêt définis sur des variables) ou le type d'exception (pour les points d'arrêt définis sur les exceptions).

Rendre les Points d'Arrêt Conditionnels

Vous pouvez définir un point d'arrêt pour seulement suspendre l'exécution du code lorsqu'une condition donnée est remplis. Par exemple, si vous avez une longue boucle For et que vous désirez voir ce qui se produit juste avant que la boucle ne finisse, vous pouvez lier le point d'arrêt au fait que l'itérateur atteigne une valeur donnée.

Voici quelques exemples de conditions que vous pouvez placer sur un point d'arrêt:

- `i==4` (ce qui signifie que l'exécution ne sera suspendue à ce point d'arrêt que lorsque la variable `i` vaudra 4)
- *`ObjectVariable!`*`=null` (ce qui signifie que l'exéction ne sera suspendue à ce point d'arrêt que lorsque *`ObjectVariable`* aura une valeur assignée)
- *`NomMéthode`* (où *`Method`* a un type de retour booléen. L'exécution ne sera suspendue à ce point d'arrêt que lorsque la `Méthode` retournera `true`.
- *`CollectionX`*`.contains(`*`ObjectX`*`)` (ce qui signifie que l'exécution ne devrait s'arrêter au point d'arrêt que si *`ObjectX`* était dans la collection.

Pour rendre un point d'arrêt conditionnel:

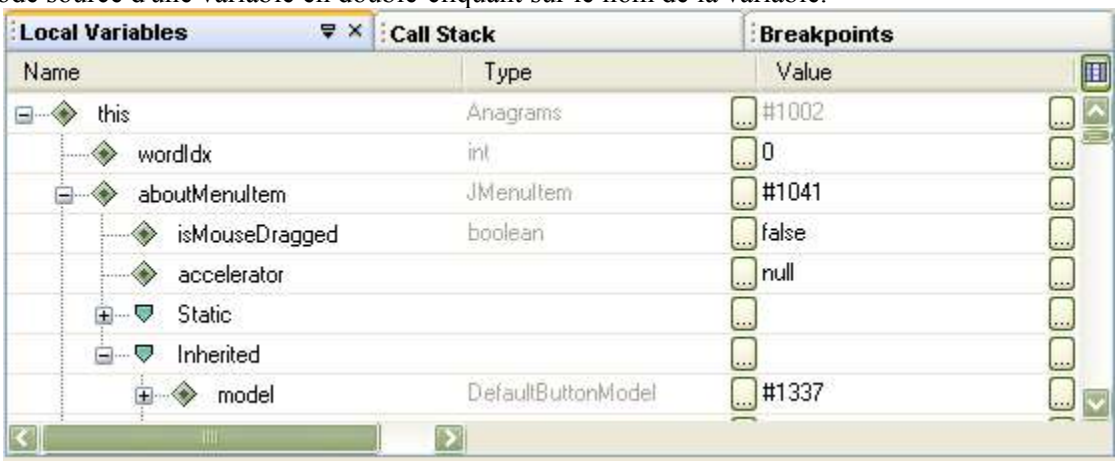
1. Ouvrez la fenêtre Breakpoints en pressant Maj-Alt-5.
2. Dans la fenêtre Breakpoints, cliquez-droit sur le point d'arrêt pour lequel vous désirez placer une condition et choisissez Customize.

3. Dans la boite de dialogue Customize Breakpoint, remplissez le champ Condition avec la condition qui devra être satisfaite pour suspendre l'exécution à ce point d'arrêt.

Les points d'arrêt conditionnels sont marqués par l'icone .

Travailler avec des Variables et Expressions

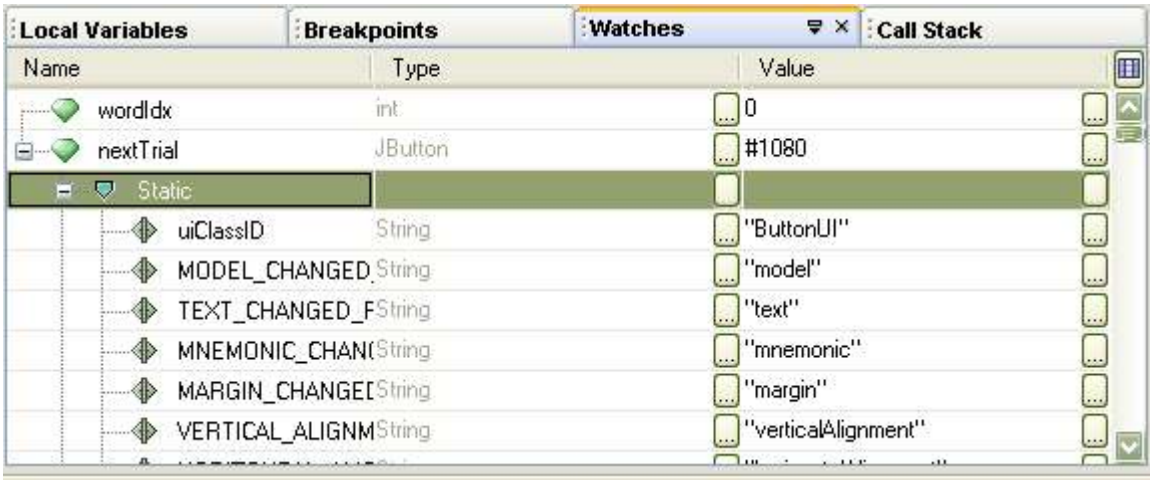
Lorsque vous avancez dans un programme, vous pouvez surveiller les valeurs des champs et variables locales. La fenêtre Local Variables affiche toute les variables qui sont actuellement connues dans le contexte actuel de l'exécution du programme et fournit une liste de leur type et valeur. Si la valeur de variable est une référence d'objet, la valeur est donnée avec un le signe cardinal (#) et un nombre qui sert d'identifiant à l'instance de l'objet. Vous pouvez aller dans le code source d'une variable en double-cliquant sur le nom de la variable.



Vous pouvez également créer une vue plus personnalisée des variables et expression en définissant des surveillances et les visualisant dans la fenêtre Watches.

La fenêtre Watches (Maj-Alt-2) differe de la fenêtre Local Variables sur les points suivants:

- La fenêtre Watches ne montre que les valeurs des variables ou expressions que vous avez spécifiées, ce qui perment de ne pas avoir la fenêtre trop encombrée.
- La fenêtre Watches affiche toute les surveillances que vous avez défini, que les variables soient dans le contexte ou non. Si la variable existe séparément dans différents contextes, la valeur donnée dans la fenêtre Watches s'applique à la valeur dans le contexte actuel (pas nécessairement dans le contexte dans lequel il a été défini).
- Les Surveillances persistent durant les sessions de débogage.



Définir une Surveillance sur une Variable ou un Champs

Pour définir une surveillance sur une variable ou une expression, cliquez-droit sur la variable dans l'Editeur de Source et choisissez New Watch. La variable ou expression est alors rajoutée dans la fenêtre Watches.

> **Conseil EDI NetBeans**
>
> Lorsque vous déboguez, vous pouvez également vérifier la valeur d'une variable dans le context actuel de débogage en plaçant le curseur de la souris sur la variable dans l'Editeur de Source pour afficher un tool tip avec la valeur.

Surveiller l'Objet Assigné à une Variable

Vous pouvez créer un dénommé *fixed watch* pour surveiller un objet qui est assigné à une variable (plutôt que la valeur de la variable elle-même).

Pour créer un *fixed watch*:

1. Après avoir démarrer une session de débogage, ouvrez la fenêtre Local Variables (Maj-Alt-1).
2. Cliquez-droit sur la variable pour laquelle vous aimeriez créer un *fixed watch* et choisissez Create Fixed Watch.

Un *fixed watch* est alors rajouté dans la fenêtre Watches avec l'icône . Du fait que le *fixed watch* s'applique à une instance d'objet spécifique créée durant la session de débogage, le *fixed watch* est enlevé lorsque la session de débogage est terminée.

Afficher la Valeur de la Méthode toString d'une Classe

Vous pouvez ajouter une colonne aux fenêtres Local Variables et Watches pour afficher le résultat de la méthode `toString` d'un objet. Cela vous permet d'avoir des informations plus utiles (comme les valeurs des champs actuellement assignées) sur un objet que l'identifiant numérique que la colonne Value fournit.

Pour afficher la colonne toString() dans l'une de ces fenêtres:

1. Ouvrez la fenêtre Local Variables (Maj-Alt-1) ou Watches (Maj-Alt-2).
2. Cliquez sur le bouton dans le coin supérieur droit de la fenêtre.
3. Dans la boite de dialogue Change Visible Columns, cochez la case `toString()`.

Conseil EDI NetBeans

Si vous ne voyez pas la colonne toString() apparaître directement, essayez de réduire la largeur de la colonne Value pour faire de la place pour que la colonne toString() puisse apparaître.

Modifier les Valeurs des Variables ou Expressions

Lorsque vous déboguez un programme, vous pouvez modifier les valeurs d'une variable ou d'une expression qui est affichée dans les fenêtres Local Variables ou Watches. Par exemple, vous pouvez augmenter la valeur d'un itérateur pour vous rendre plus rapidement à la fin de la boucle.

Pour modifier la valeur d'une variable:

1. Ouvrir la fenêtre Watches ou Local Variables.
2. Dans le champs Value de la variable ou expression, introduisez la nouvelle valeur et pressez la touche d'Entrée.

Afficher des Variables Des Appels de Méthodes Précédents

La fenêtre Call Stack affiche tous les appels de la chaine actuelle. Si vous désirez voir le statut des variables d'une autre appel dans la chaine, vous pouvez ouvrir la fenêtre Call Stack (Maj-Alt-3), cliquez-droit sur le noeud de la méthode et choisissez Make Current.

Vous pouvez également naviguer à travers les éléments de la pile d'appel en utilisant les commandes Make Callee Current (Ctrl-Alt-Haut) et Make Caller Current (Ctrl-Alt-Bas).

Indiquez une autre méthode courante ne modifie pas le marqueur du programme. Si vous continuez l'exécution du programme avec une des commandes de pas à pas ou la commande Continue, le programme va reprendre à partir de l'endroit om l'exécution fut suspendue.

Revenir d'une Méthode à son Appel

Sous certaines circonstances, il peut être utile de pouvoir revenir en arrière dans le code. Par exemple, si vous avez atteint un point d'arrêt et que vous désirez voir comment fonctionne le code qui conduit jusqu'au point d'arrêt, vous pouvez enlever (“pop”) l'appel actuel de la pile d'appel pour ré-exécuter la méthode.

Vous pouvez également ouvrir la fenêtre Call Stack pour voir tous les appels de méthodes appartenant à la chaine en cours pour le thread actuel. L'appel actuel est marqué par l'icône . Les autres appels dans la pile sont marqués par l'icône .

Pour revenir à l'appel de méthode précédent:

1. Ouvrez la fenêtre Call Stack (Maj-Alt-3).
2. Cliquez-droit sur la ligne dans la fenêtre Call Stack qui représente l'endroit où vous désirez revenir et choisissez Pop to Here.

Le marqueur du programme retourne à la ligne où la méthode fut appelée. Vous pouvez alors ré-exécuter la méthode.

Pour revenir à l'appel de méthode le plus récent, vous pouvez également choisir Run | Stack | Pop Topmost Call.

Conseil EDI NetBeans

Lorsque vous revenez en arrière d'un appel, les effets du code exécuté précédemment n'est pas annulé. Ré-exécuter le code peut amener le programme à se comporter différement que lors de son exécution normale.

Supervision et Contrôle de l'Exécution de Threads

La fenêtre Threads de l'EDI (Maj-Alt-7) vous permet de voir le statut des threads du programme actuellement débogué. Il vous permet également de modifier le thread qui est actuellement supervisé dans d'autres fenêtres du débogueur (comme Pile d'Appel ou Variables Locales) et de suspendre des threads individuellement.

Modifier le thread actuel n'affecte en rien la façon dont s'exécute le programme.

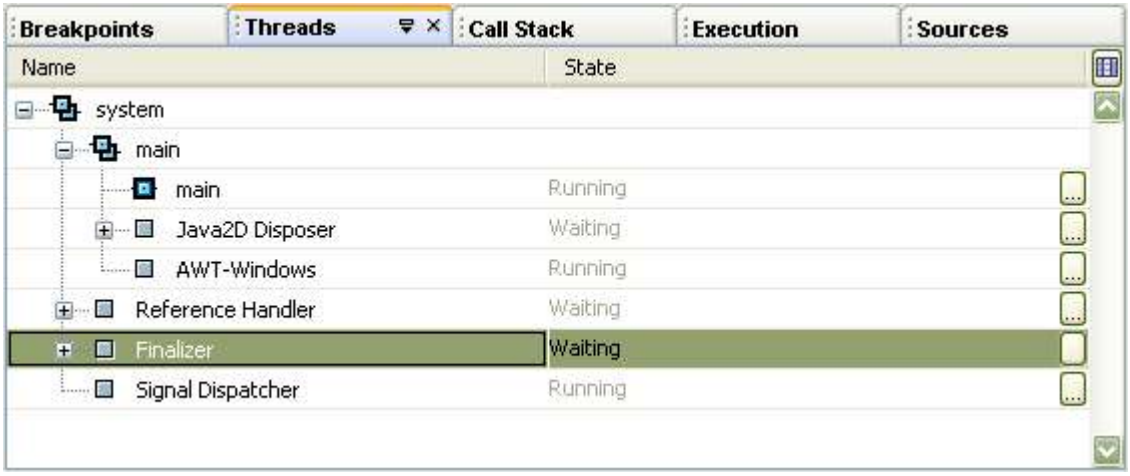


Tableau 7: Noeuds dans la fenêtre Threads

Icône	Signification
	Thread actuellement supervisé
	Groupe de threads actuellement supervisé
	Thread en cours d'exéction
	Thread suspendu
	Groupe de Threads

Modifier le Thread Actuellement Supervisé

Le contenu des fenêtres Pile d'Appel et Variables Locales est dépendant du thread en cours de supervision par le débogueur (également nommé *current thread*). Pour changer de thread actuellement supervisé:

1. Ouvrez la fenêtre Threads en pressant Maj-Alt-7.
2. Cliquez-droit sur le thread que vous désirez superviser et choisissez Make Current.

Suspendre et Continuer les Threads

Vous pouvez suspendre l'exécution d'un thread en cliquant-droit sur son noeud dans la fenêtre Threads et en choisissant Suspend. Vous pouvez continuer l'exécution d'un thread suspendu en cliquant-droit sur son noeud et choisissant Resume.

Suspendre un Simple Thread à un Point d'Arrêt

Par défaut, lorsqu'un programme atteint un point d'arrêt, tous les threads sont suspendus. Cependant, vous pouvez également configurer un point d'arrêt pour que ce ne soit que son thread qui soit suspendu lorsque le point d'arrêt est atteint:

1. Ouvrez la fenêtre Breakpoints en pressant Maj-Alt-5.
2. Dans la fenêtre Breakpoints, cliquez-droit sur le point d'arrêt et choisssez Customize.
3. Dans la boite de dialogue Personaliser le Point d'Arrêt, sélectionnez Current depuis la liste dérourante Suspend.

Isoler le Débogage à un Simple Thread

Par défaut, tous les threads de l'application sont exécutées dans le débogueur. Si vous désirez isoler le débogage pour qu'il n'y ait qu'un seul thread dans le débogueur:

1. Soyez sûr que le thread que vous déboguez est désigné comme le thread actuel dans la fenêtre Threads (Maj-Alt-7). Le thread actuel est marqué avec l'icône .
2. Ouvrez la fenêtre Session en pressant Maj-Alt-6.
3. Dans la fenêtre Sessions, cliquez-droit sur le noeud de la session et choisissez Scope | Debug Current Thread.

Fixer du Code Durant une Session de Débogage

En utilisant la fonctionnalité Fix de l'EDI, il est possible d'affiner les réglages du code au milieur d'une session de débogage et de continuer le débogage sans avoir à redémarrer une nouvelle session de débogage. Cela peut vous faire gagner beaucoup de temps que vous auriez autrement perdu à attendre que vos sources soient recompilées et la session de débogage redémarrée.

La fonctionnalité de Fix est très utile pour des situations où vous avez besoin de :

- Retravailler l'apparence d'un composant visuel que vous avez créé.
- Modifier la logique dans une méthode.

La fonctionnalité de Fix ne fonctionnera pas si vous devez faire l'une des choses suivantes durant la session de débogage:

- Ajouter ou supprimer des méthodes ou champs.
- Modifier les accès d'une classe, d'un champ, ou d'une méthode.
- Refactoriser la hiérarchie de classe.
- Modifier du code qui n'a pas encore été chargé dans la machine virtuelle.

Pour utiliser la commande Fix tout en déboguant:

1. Lorsque l'exécution est suspendue durant une session de debogage, faites toutes les corrections nécessaires dans l'Editeur de Source.
2. Choisissez Run | Fix pour recompiler le fichier et rendre la classe recompilée disponible au débogueur.
3. Charger le code fixé dans le débogueur.

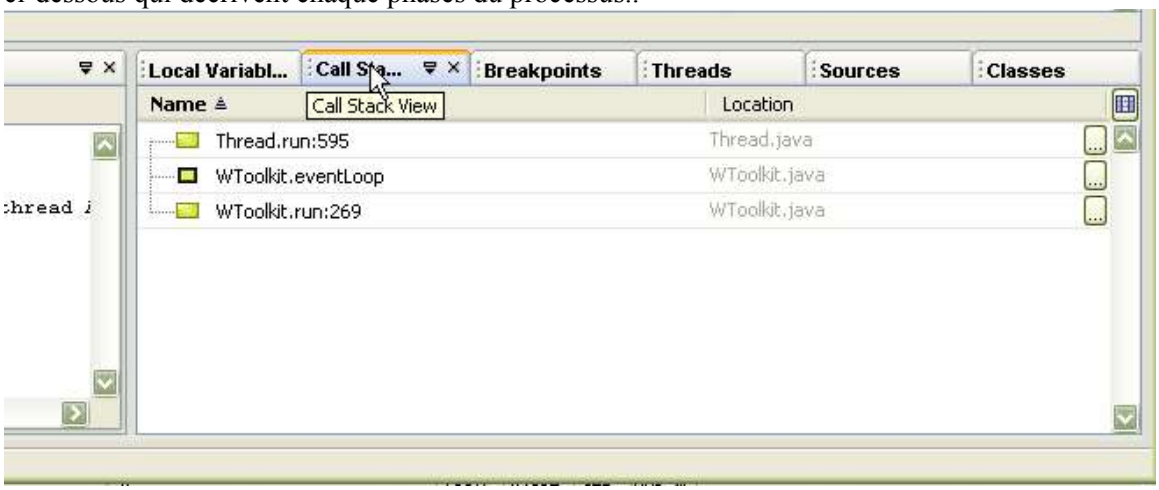
   Si vous avez modifié la méthode actuelle, cela est fait automatiquement. La méthode est automatiquement “retirée” de la pile d'appel, signifiant que le marqueur de programme retourne à la ligne où la méthode était appelée. Vous pouvez alors exécuter le code modifié en avançant pas à pas dans la méthode (F7) ou en avant jusqu'après l'appel de la méthode (F8).

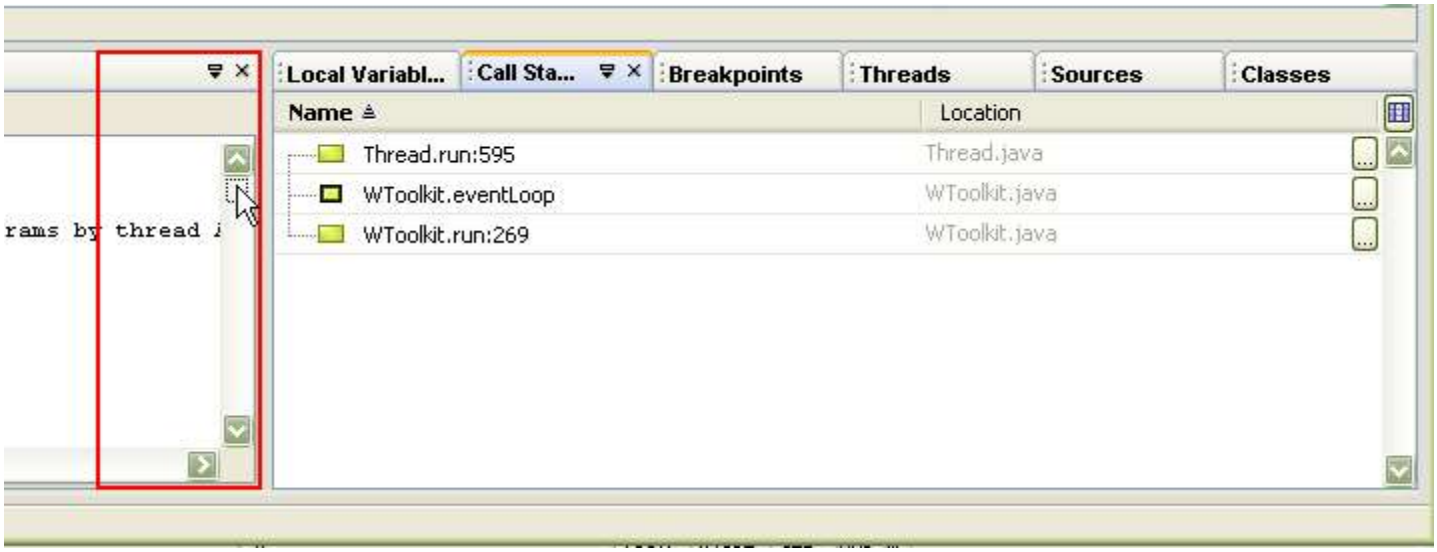
   Pour un élément Graphique comme un composant JDialog, vous pouvez fermer le composant (ou le containeur du composant) et le ré-ouvrir pour charger le code fixé dans le débogueur.
4. Répéter les étapes 1-3 si nécessaire.

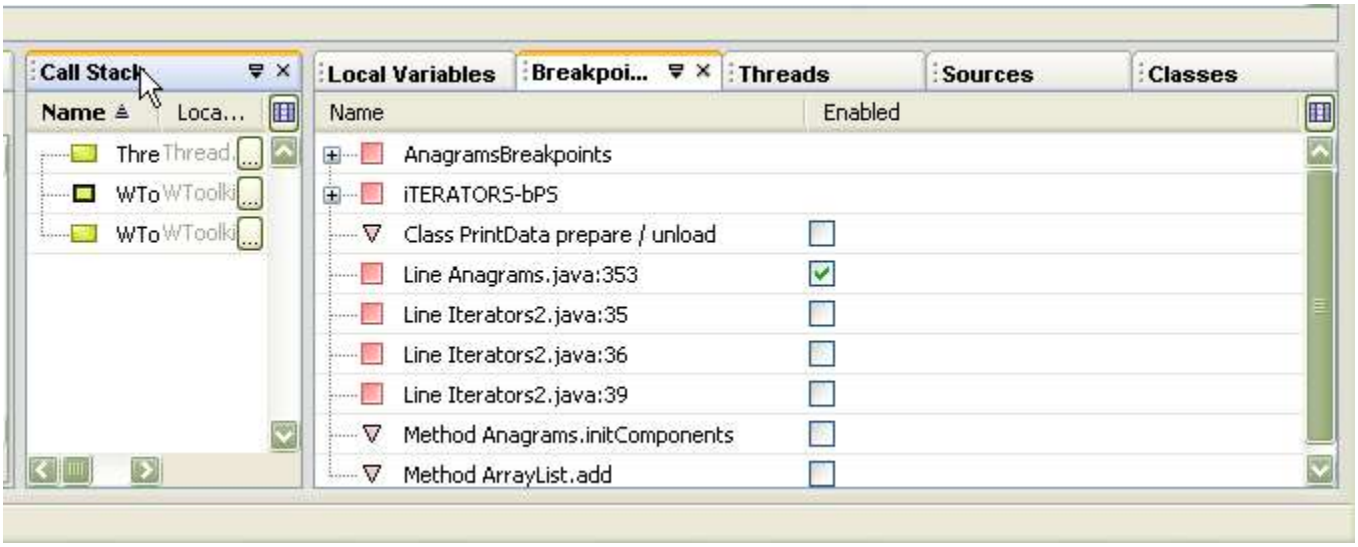
Voir Plusieurs Fenêtres de Débogage Simultanément

Par défaut, les fenêtres du débogueur apparaissent dans des onglets dans le coin inférieur droit de l'EDI. Seul l'un des onglets est visible à la fois. Si vous désirez voir plusieurs fenêtres en même temps, vous pouvez utiliser le “drag and drop” (tirer-lâcher) pour mettre un onglet dans sa propre fenêtre ou le déplacer vers une autre fenêtre (celle occupée par la console du débogueur par exemple). Vous pouvez également déplacer le séparateur entre les fenêtres pour modifier la taille de chaque fenêtre.

Pour créer une fenêtre séparée pour un onglet du débogueur, cliquez sur l'onglet désiré et déplacer le curseur de souris jusqu'à ce qu'un encadré rouge apparaisse à l'endroit où la nouvelle fenêtre devrait apparaître. Ensuite, lâchez le bouton de la souris. Voyez les trois copies d'écran ci-dessous qui décrivent chaque phases du processus..




Local Variabl...
Call Sta...
Breakpoints
Threads
Sources
Classes
Name
Location
Thread.run:595
Thread.java
WToolkit.eventLoop
WToolkit.java
WToolkit.run:269
WToolkit.java


Call Stack
Local Variables
Breakpoi...
Threads
Sources
Classes
Name
Loca...
Name
Enabled
AnagramsBreakpoints
iTERATORS-bPS
Class PrintData prepare / unload
Line Anagrams.java:353
Line Iterators2.java:35
Line Iterators2.java:36
Line Iterators2.java:39
Method Anagrams.initComponents
Method ArrayList.add

MCours.com