JAVA – HOWTO 4
Débugger

Université de Mons-Hainaut

Il existe plusieurs stratégies que vous pouvez utiliser pour reconnaître des bugs et leur causes.

Etape 1 *Reproduire l'erreur*

En testant votre programme, vous avez remarqué que quelque chose est erroné : il donne un mauvais résultat, il semble imprimer des choses aléatoires, il entre dans une boucle infinie, ou il "plante". Essayez de *reproduire* exactement ce comportement. Quels chiffres avez-vous entrés ? Où avez-vous cliqué ?

Exécutez le programme une nouvelle fois; entrez exactement les mêmes données et cliquez sur les mêmes boutons. Le programme reproduit-il le comportement erronné ? Si oui, utilisez un débugueur ou tracez le programme pour analyser ce problème particulier.

Etape 2 *Simplifier l'erreur*

Avant de lancer le débugueur ou de tracer votre programme, passez quelques minutes pour essayer de trouver des données ou un comportement plus simples et qui produisent également une erreur. Pouvez-vous utilisez des mots plus courts ou des nombres plus simples en ayant toujours un problème ? Si oui, le débugage en sera simplifié.

Etape 3 *Diviser pour règner*

Maintenant que vous avez détecté une erreur particulière, il faut vous en "approcher" le plus près possible. La clé du débugage est de localiser le morceau de code qui produit l'erreur. Trouvez un bug est en général plus difficile que de le corriger. Supposez que votre programme se plante avec une division par zéro. Comme il y a de nombreuses divisions dans un programme type, il n'est souvent pas possible de les observer toutes (par l'intérmédiaire du débugueur ou en affichant des messages). A la place, utilisez la technique *diviser pour règner*. A partir du code de la méthode `main`, détectez dans quelle instruction apparaît le bug (faites un *step over* pour chaque méthode dans un débugueur ou affichez les valeurs critiques à certains endroits clefs dans le cas du traçage du programme). Si le bug semble subvenir lors d'un appel d'une méthode particulière, continuez votre inspection à partir de cette méthode et répétez le processus.

Etape 4 *Savoir ce que votre programme doit faire*

La trace de votre programme (ou le débugueur) vous montre ce que celui-ci *fait* en réalité. Vous devez savoir ce que votre programme *devrait faire*, sinon vous ne serez pas capable de trouver les bugs. Avant de tracer une boucle, demandez-vous combien d'itérations celle-ci est supposée réaliser. Avant d'inspecter une variable, demandez-vous ce qu'elle devrait contenir. Si vous n'avez aucun indice, prenez d'abord le temps d'y réfléchir. Ayez une calculatrice (ou un programme indépendant) pour refaire les calculs. Quand vous savez ce que la variable devrait contenir, commencez l'inspection. C'est le moment de vérité : si le programme est toujours sur le bon chemin (la valeur de la variable est celle qui était supposée), vous devez chercher le bug plus loin. Si la valeur est différente, vous avez peut-être localisé le bug. Revérifiez vos calculs. Si vous êtes sûr de vous, trouvez pourquoi votre programme donne une autre valeur.

Dans bien des cas, les bugs résultent d'erreurs simples telles que les conditions de fin d'une boucle (erreur *off by one*). Cela peut être également des erreurs de calcul (un *moins* ou lieu d'un *plus*).

Etape 5 *Regarder les détails*

Quand on débugge un programme, on a souvent une théorie sur la nature du problème. Gardez cependant l'esprit ouvert et regarder tous les détails. Quels messages étranges sont affichés? Pourquoi le programme fait quelque chose d'inattendu? Ces détails comptent. Quand vous débugger un programme, mettez-vous dans la peau d'un détective qui a besoin de vérifier chaque indice.

Si vous remarquez une autre défaillance en corrigeant votre problème, il vaut mieux la régler de suite. Cette défaillance est peut-être à la base de votre problème original. Il est préferable de noter votre problème original, puis fixer ce que vous venez de découvrir, et ensuite revenir sur votre première mission.

Etape 6 *Soyez sûrs de comprendre le bug avant de le corriger*

Quand une boucle fait trop d'itérations, il est très tentant d'appliquer une solution rapide telle que soustraire 1 d'une variable pour que ce problème n'apparaisse plus. Une telle solution à la va-vite à de fortes chances de mener à de nouveaux problèmes ailleurs. Vous devez réellement comprendre comment votre programme devrait être écrit avant de corriger un bug.

Il arrive parfois que l'on trouve bug après bug et qu'on applique des solutions les unes après les autres : le problème ne fait que se déplacer. C'est en général un symptome d'un problème plus important lié à la logique du programme. Vous devez repensez le design du programme et le réorganiser.