

Eclipseデバッグを活用する ための31のTips

近藤寛喜(@kompiro)
kompiro@gmail.com

Thanks @shuji_w6e

目的

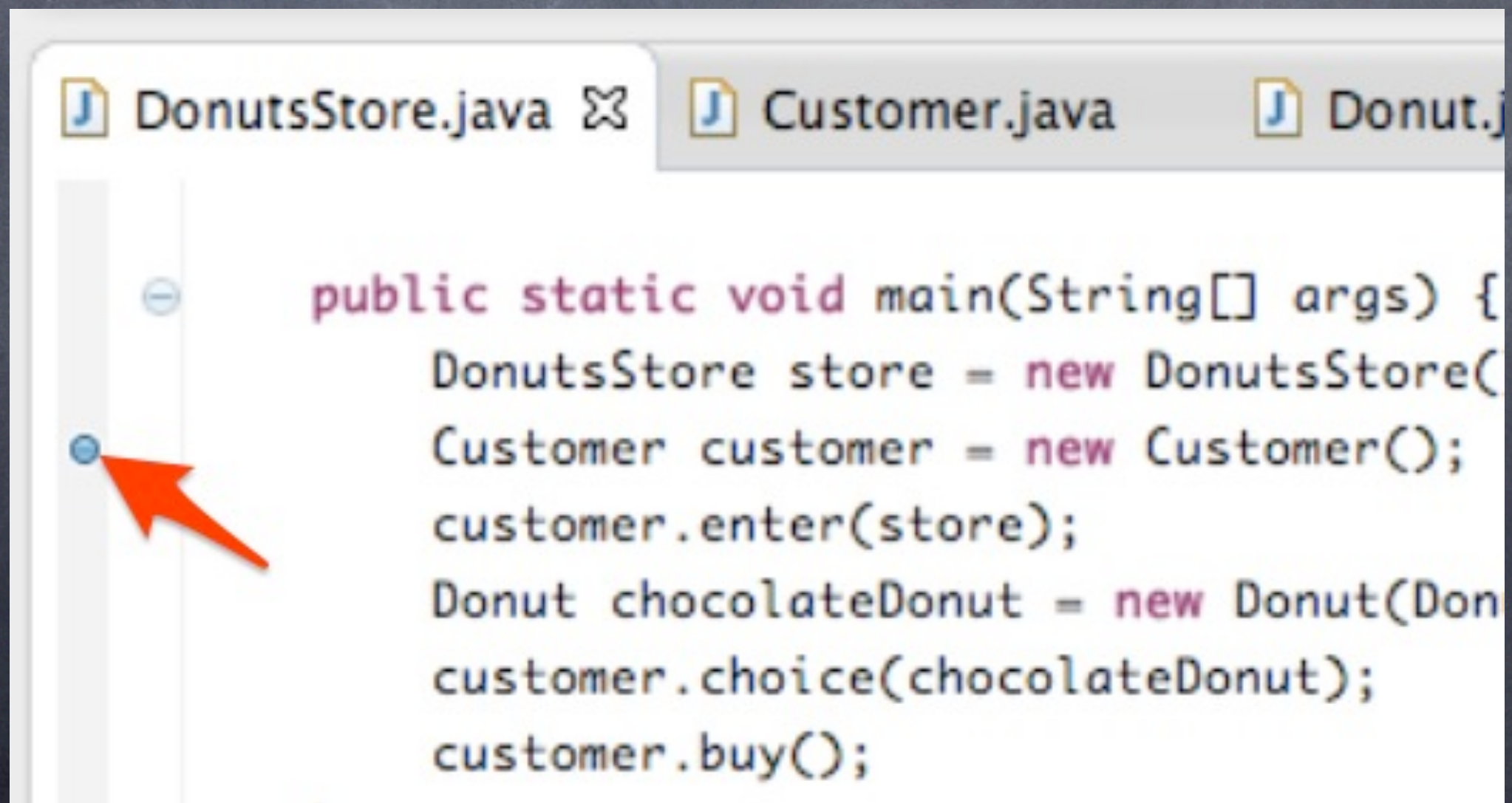
- Java開発でEclipseを使ってるエンジニアがデバッグ時に知っておくと得なTipsをまとめました。
- Q&A形式です。
- 基本編と応用編があります。
- 基本編でも、まとめている時に「これはいい」と思った機能があったので、そちらも是非目を通してくだされば。
- デバッグ例のソースコードは <https://github.com/kompiro/debug-donutstore> に公開

基本編

Q1. デバッグするにはどう
すればよいですか？

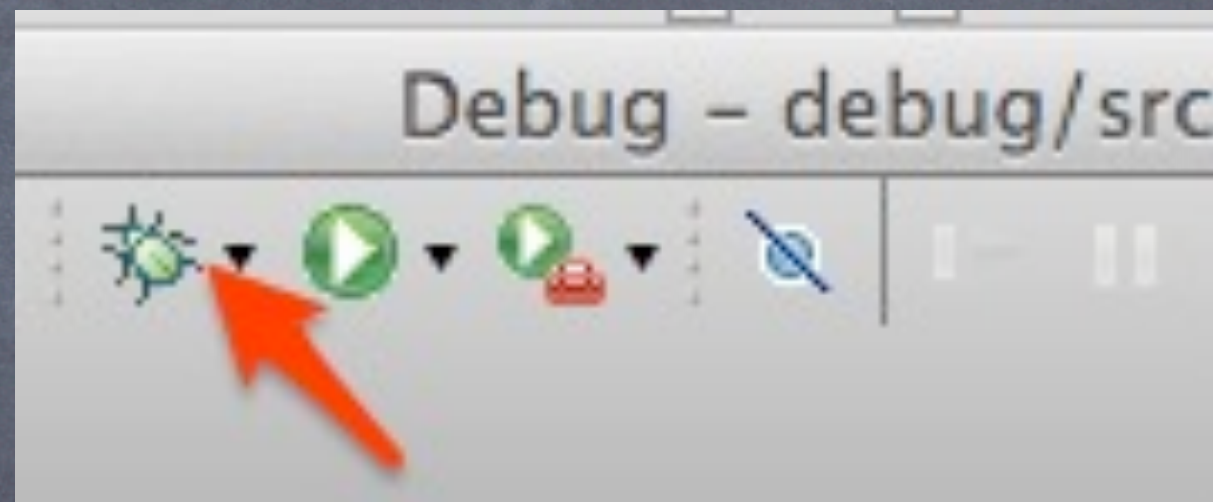
1. ブレークポイントを設定 します

設定したい行の左端をダブルクリック



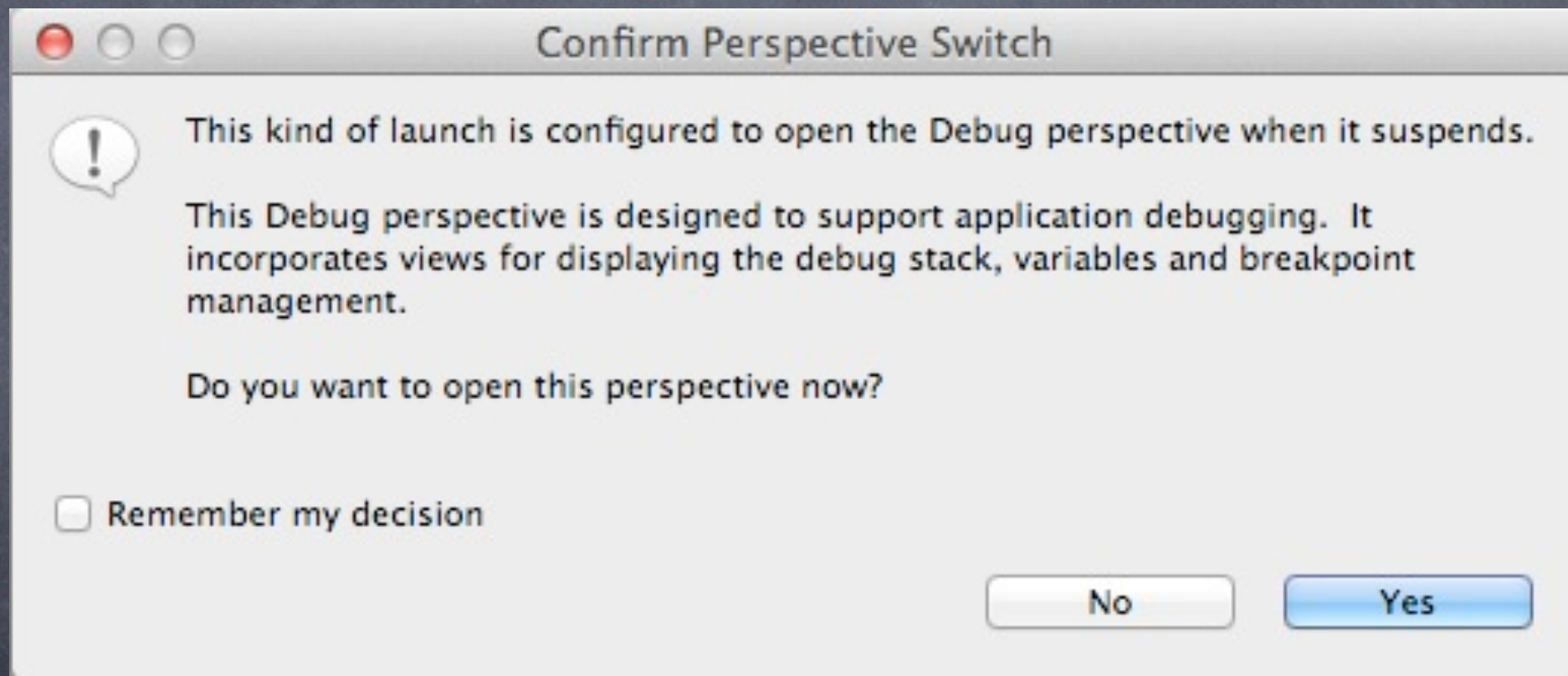
2. Debug Launcherから起動します。

ツールバー上にある虫アイコンをクリック



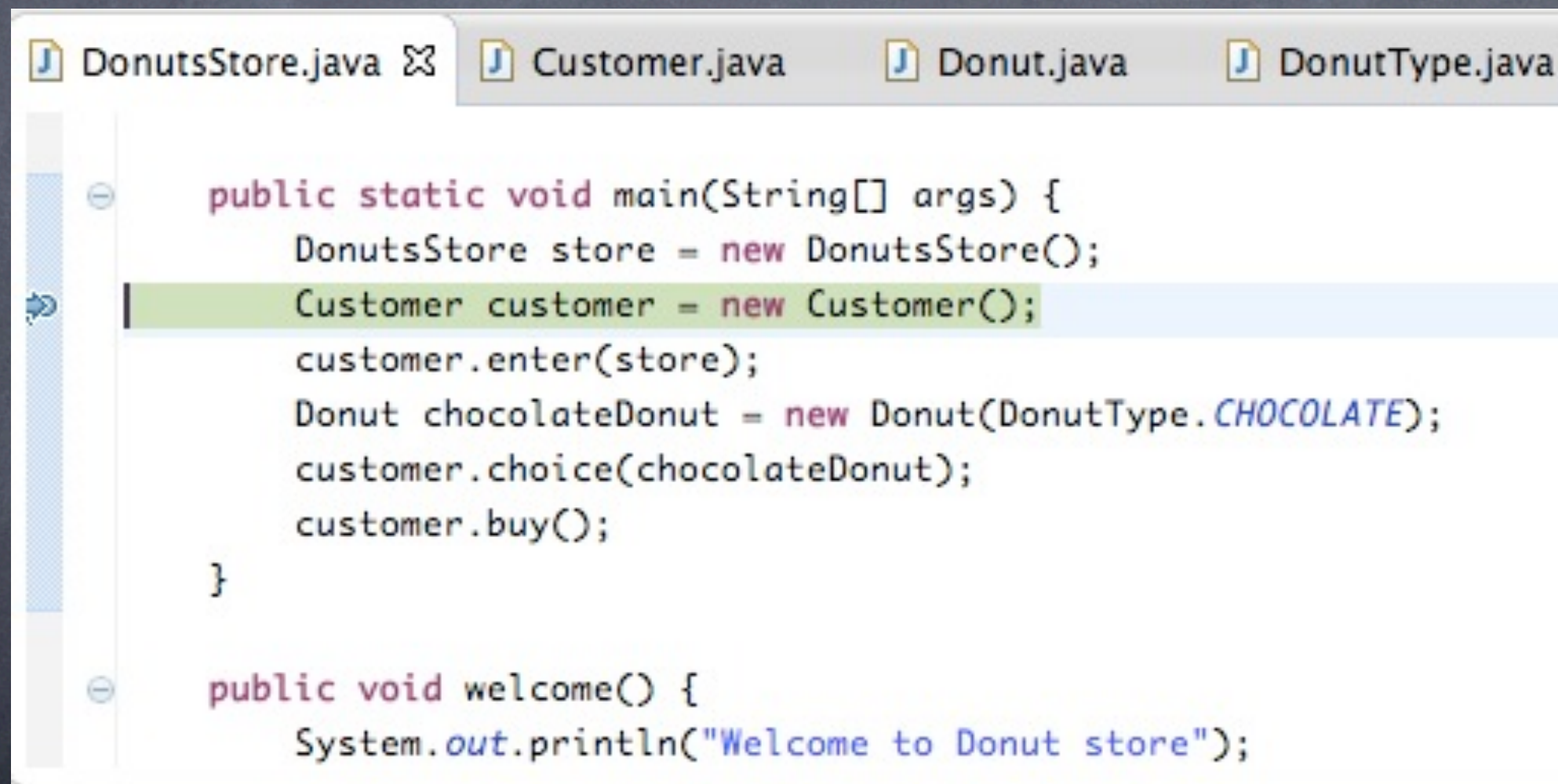
3. パースペクティブの切り替えを勧められます。

デバッグ時はデバッグ用のパースペクティブにしましょう(Yesを選んでください。)



4. 設定した場所からデバッグが開始されます。

緑色の行が、実行中の行です。



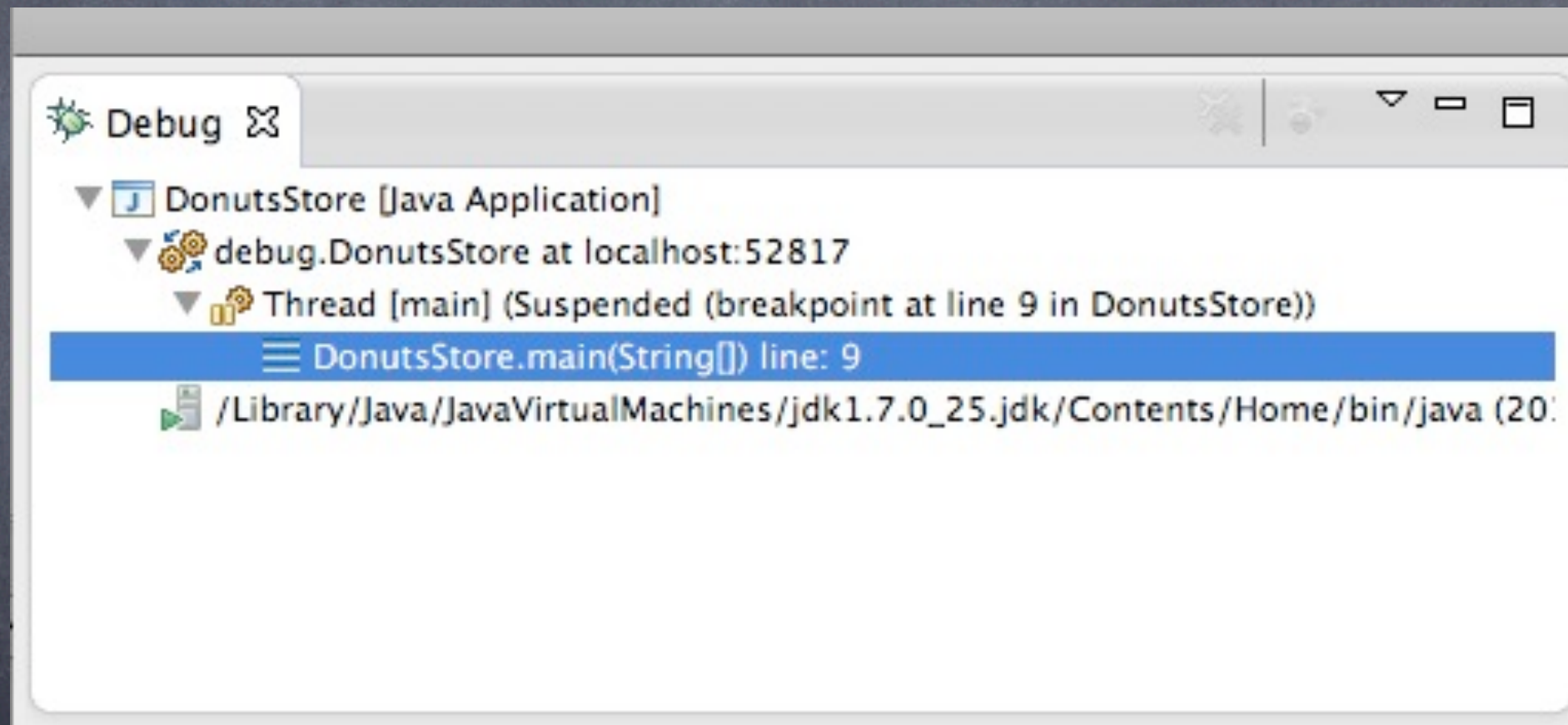
```
DonutsStore.java ⌘ Customer.java Donut.java DonutType.java

public static void main(String[] args) {
    DonutsStore store = new DonutsStore();
    Customer customer = new Customer();
    customer.enter(store);
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);
    customer.choice(chocolateDonut);
    customer.buy();
}

public void welcome() {
    System.out.println("Welcome to Donut store");
}
```


5. 設定した場所からデバッグが開始されます。

デバッグビュー(Debug)もこんな感じになります。



Q2. 毎回デバッグパースペ
クティブへの切り替えを聞
かれて鬱陶しいです。な
んとかなりませんか？

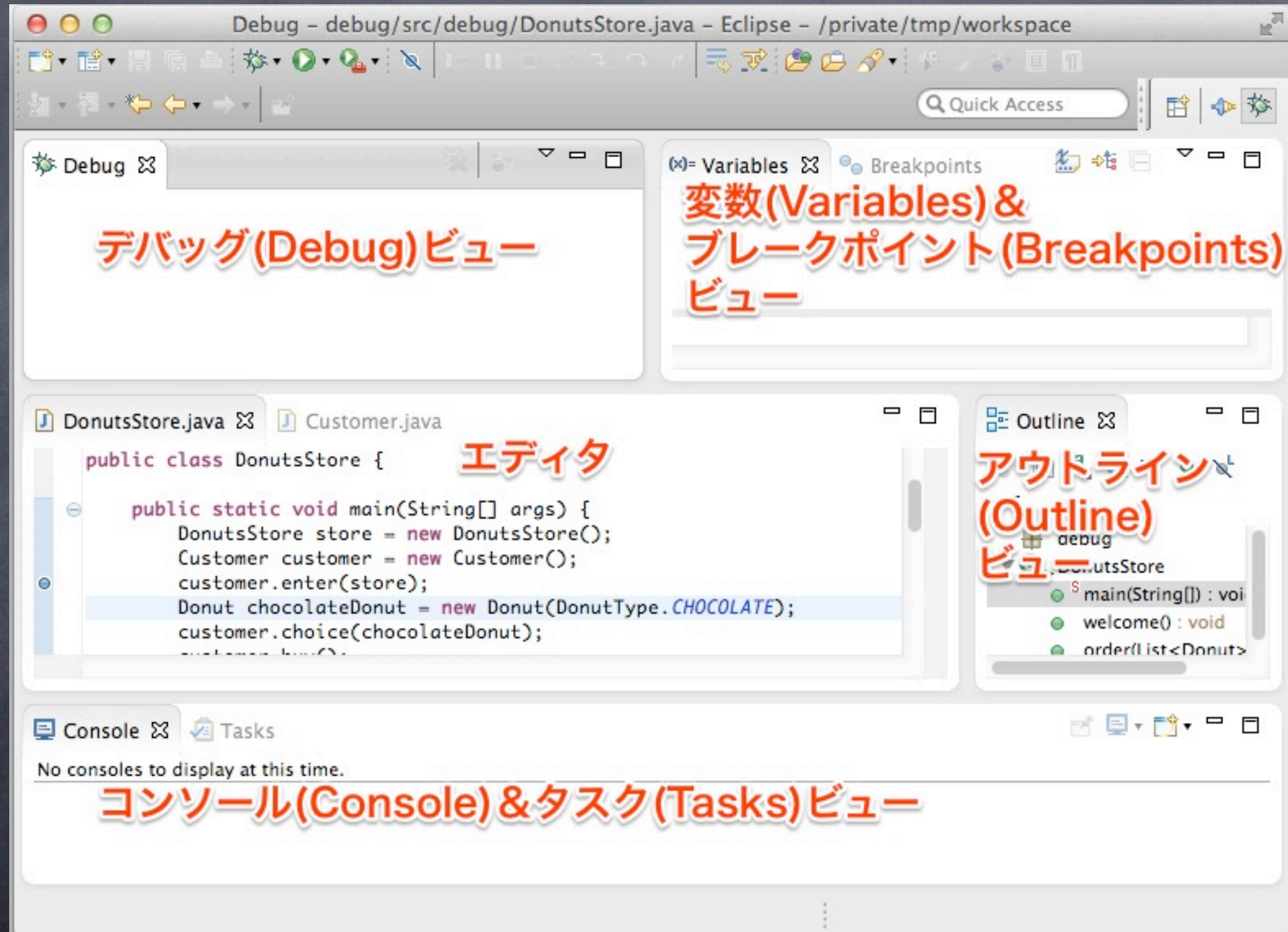
Thanks @yujiorama

A. ダイアログにチェックを入
れると出なくなります。



Q3. デバッグパースペクティブにはどんなビューがありますか？

A. 基本はこんな感じ



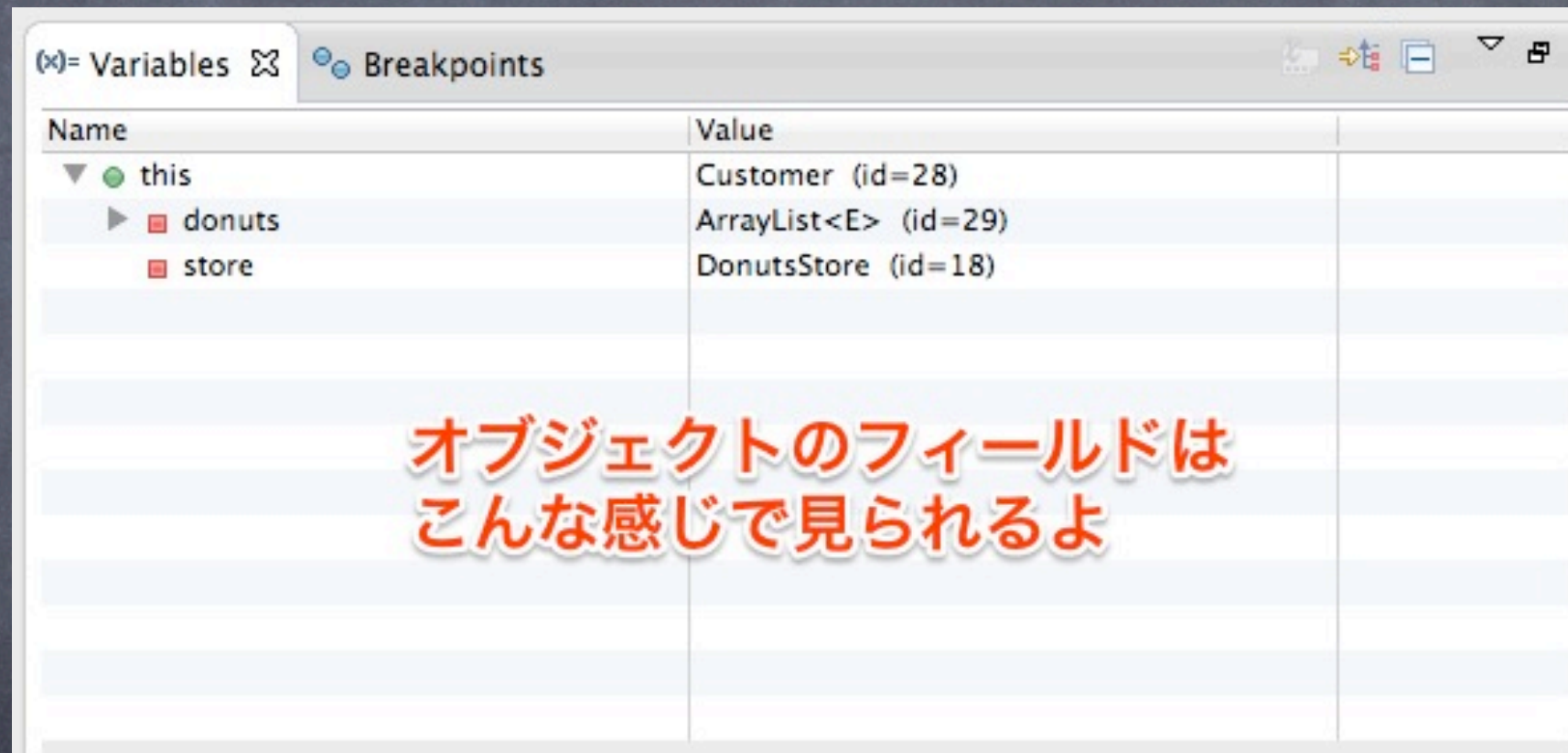
下記を追加しておくとし便利

- Display(表示)ビュー
- Expressions(式)ビュー
- この2つは後で解説します。

Q4. 実行中の変数の値を
見るにはどうすればよい
ですか？

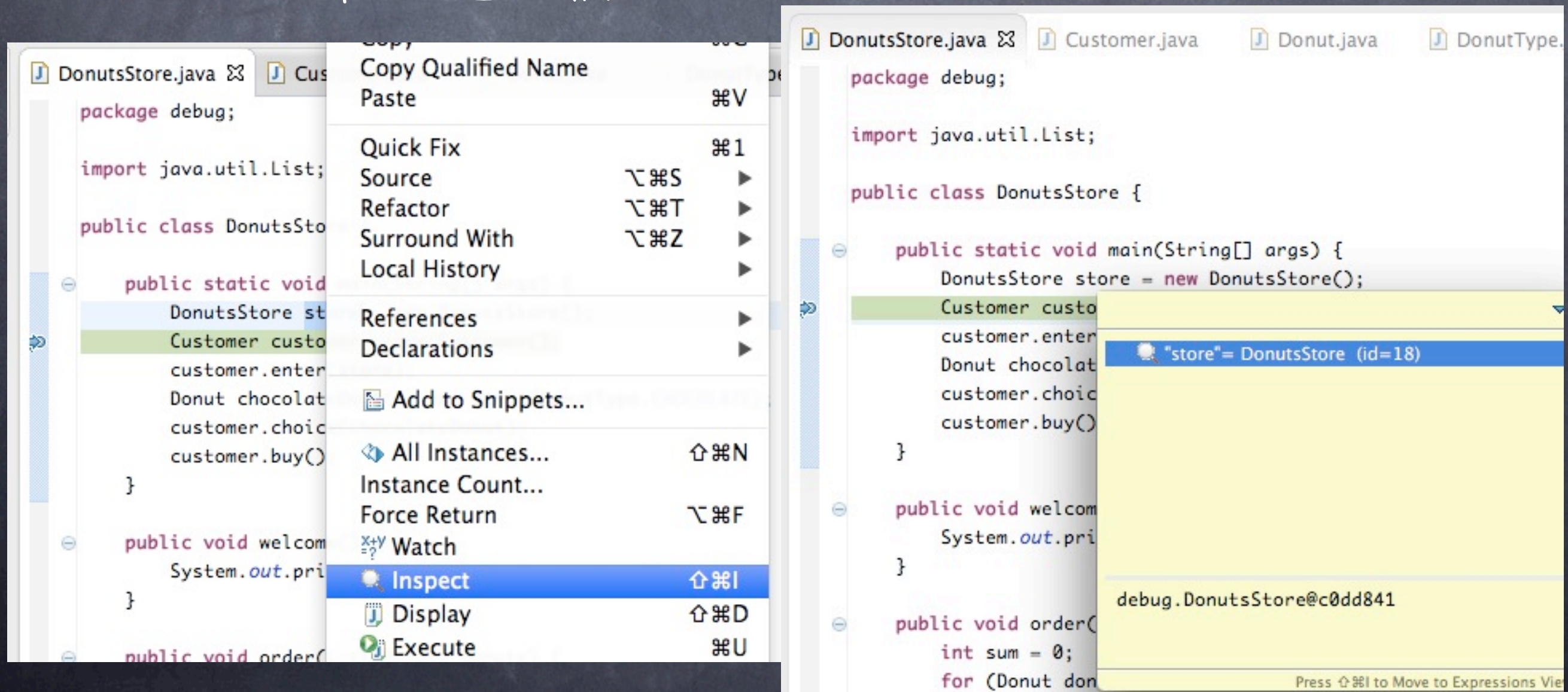
A1. 変数ビュー(Variables) で見る

変数ビューではオブジェクトの内容を見られるよ



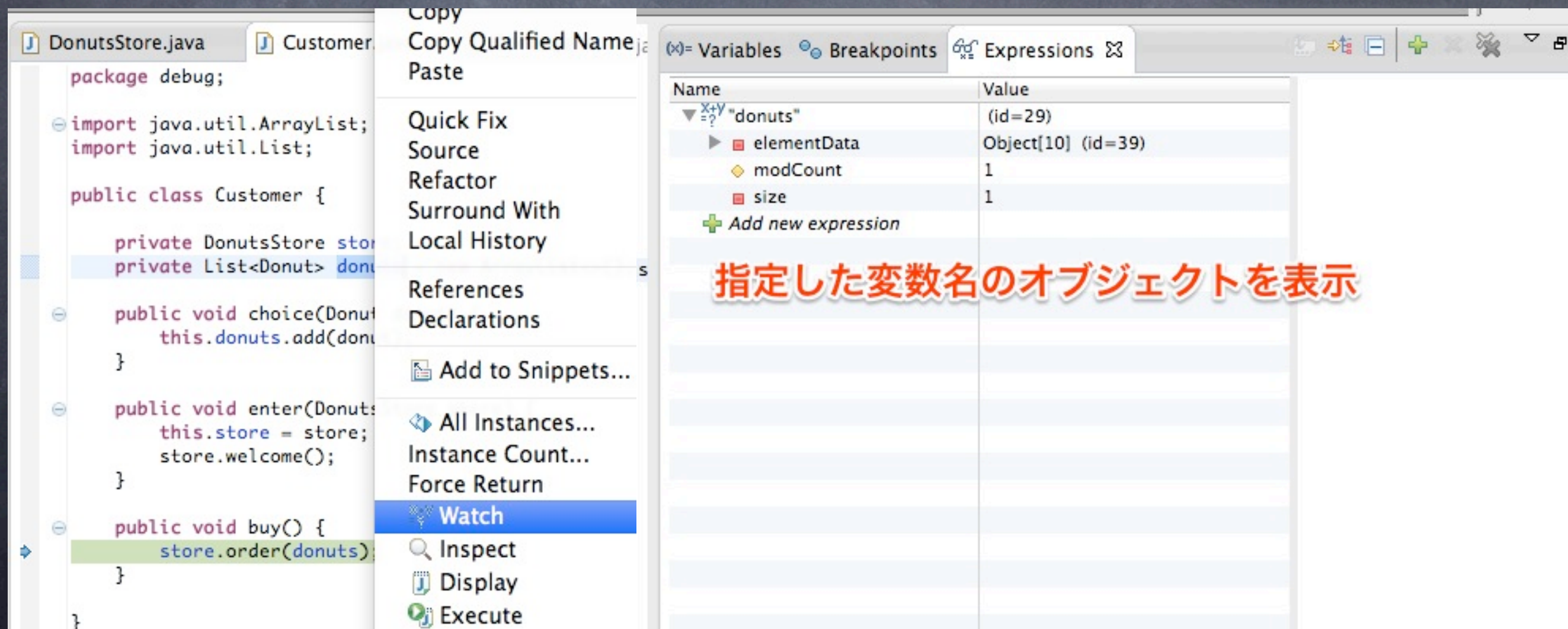
A2. Inspectを実行する

1. 見たい変数(下例:store)を選択して、右クリック
2. Inspectを選択



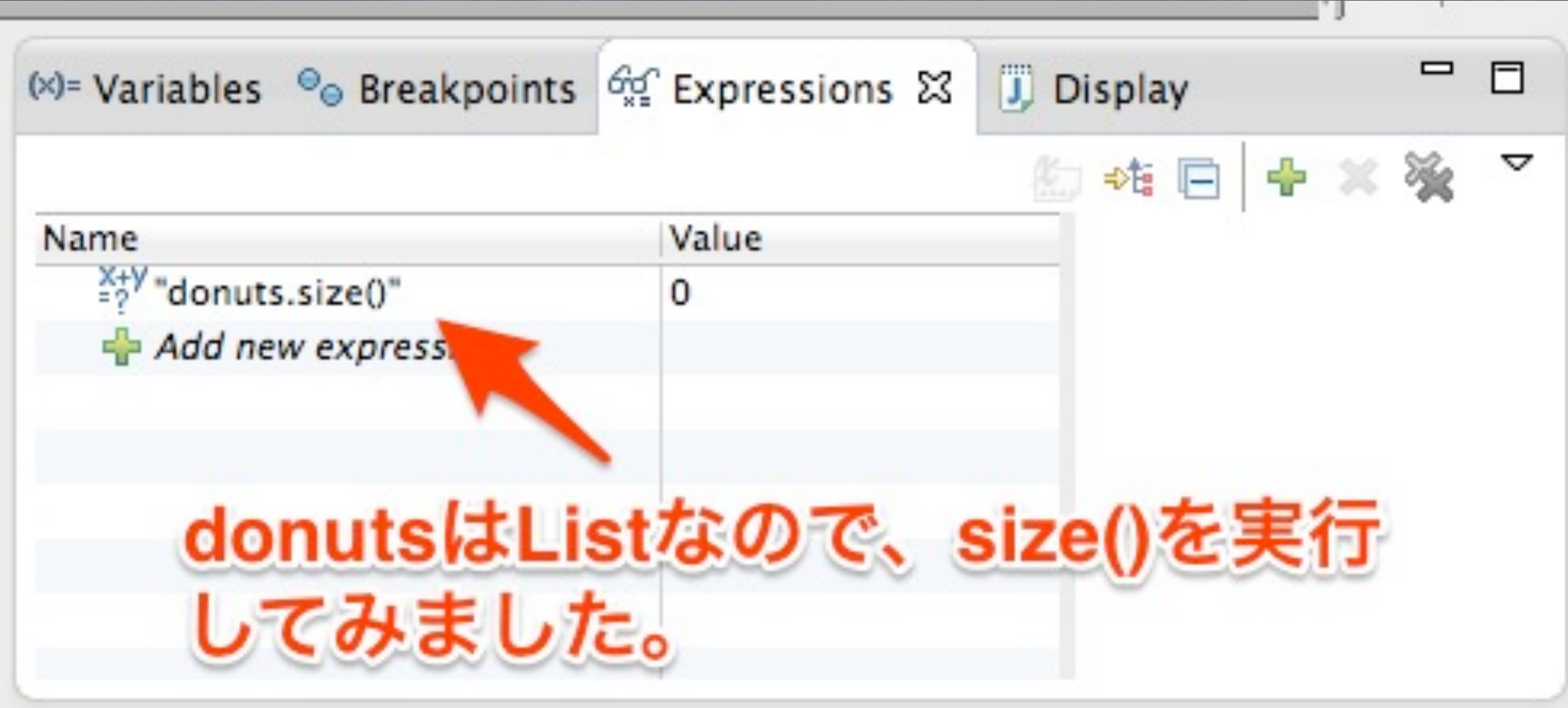
A3. 式(Expressions)ビュー —で監視

1. 監視したい変数(下例:donuts)を選択して、右クリック
2. Watchを選択



ちなみに式ビューでは

式が書ける。



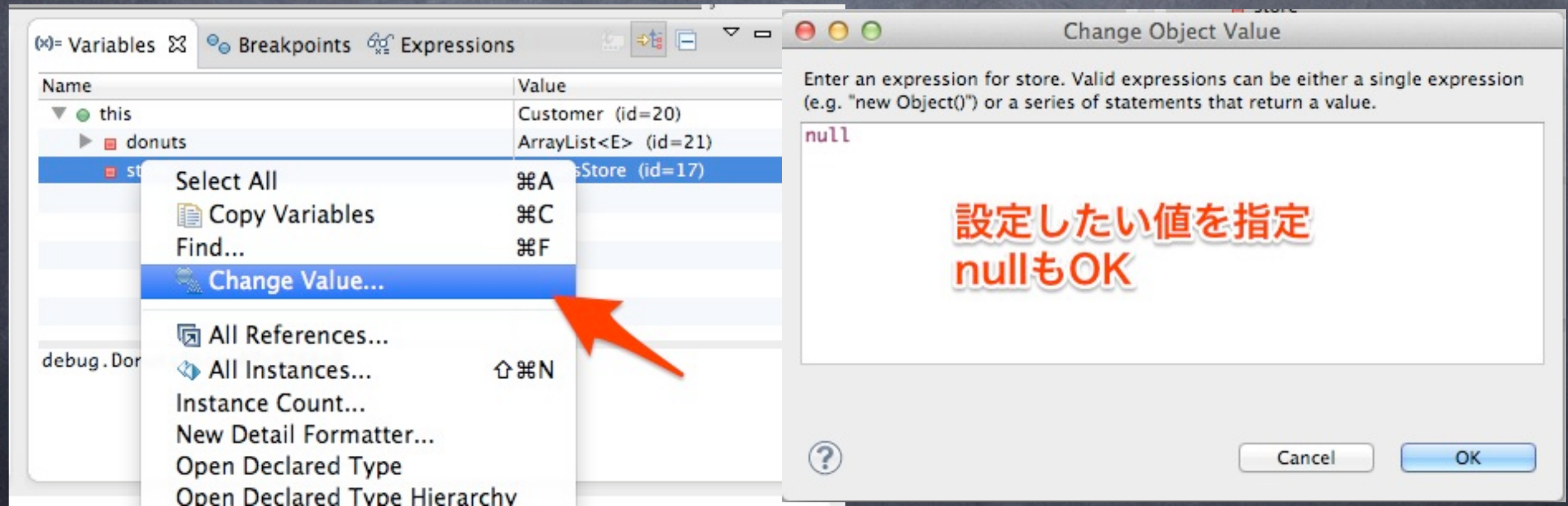
The screenshot shows the IntelliJ IDEA IDE's Expressions view. The 'Expressions' tab is active, displaying a table with two columns: 'Name' and 'Value'. The first row shows the expression `"donuts.size()"` with a value of `0`. Below this row is a button with a green plus icon and the text `Add new expression`. A red arrow points to this button. At the bottom of the screenshot, there is a red text overlay with a white outline that reads: **donutsはListなので、size()を実行してみました。**

Name	Value
<code>"donuts.size()"</code>	<code>0</code>
<code>+ Add new expression</code>	

Q5. 実行中の変数の値を
書き換えるにはどうすれば
よいですか？

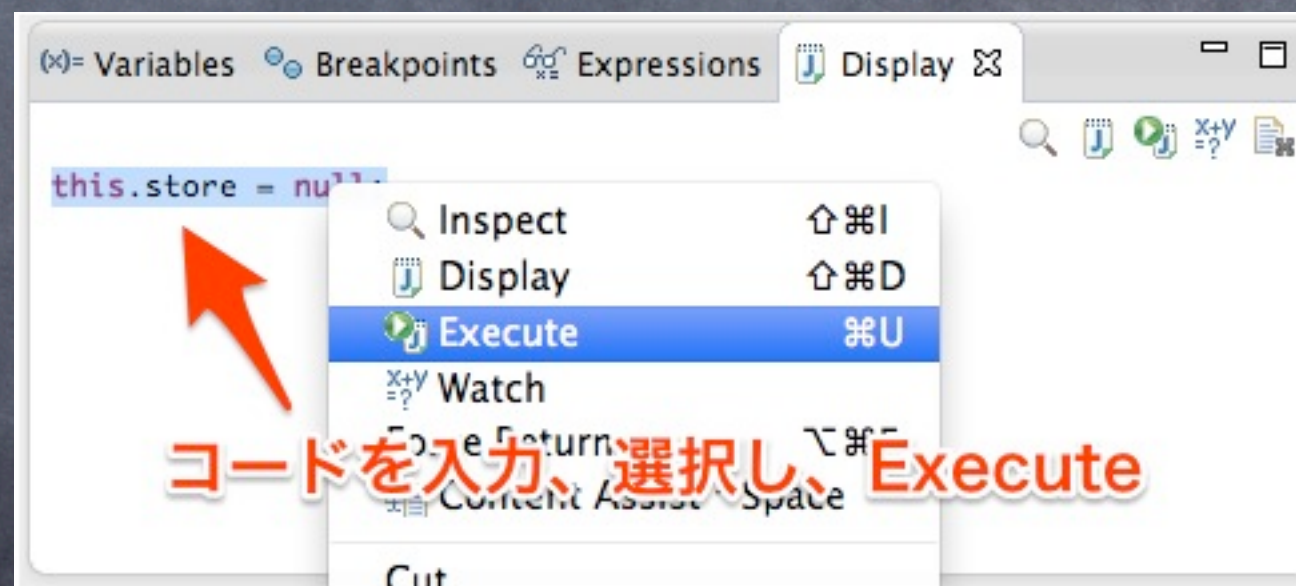
A1. 変数ビューから書き換えます。

1. 該当の変数を探し出す
2. 右クリック->Change Value...



A2. 表示(Display)ビューから書き換えます。

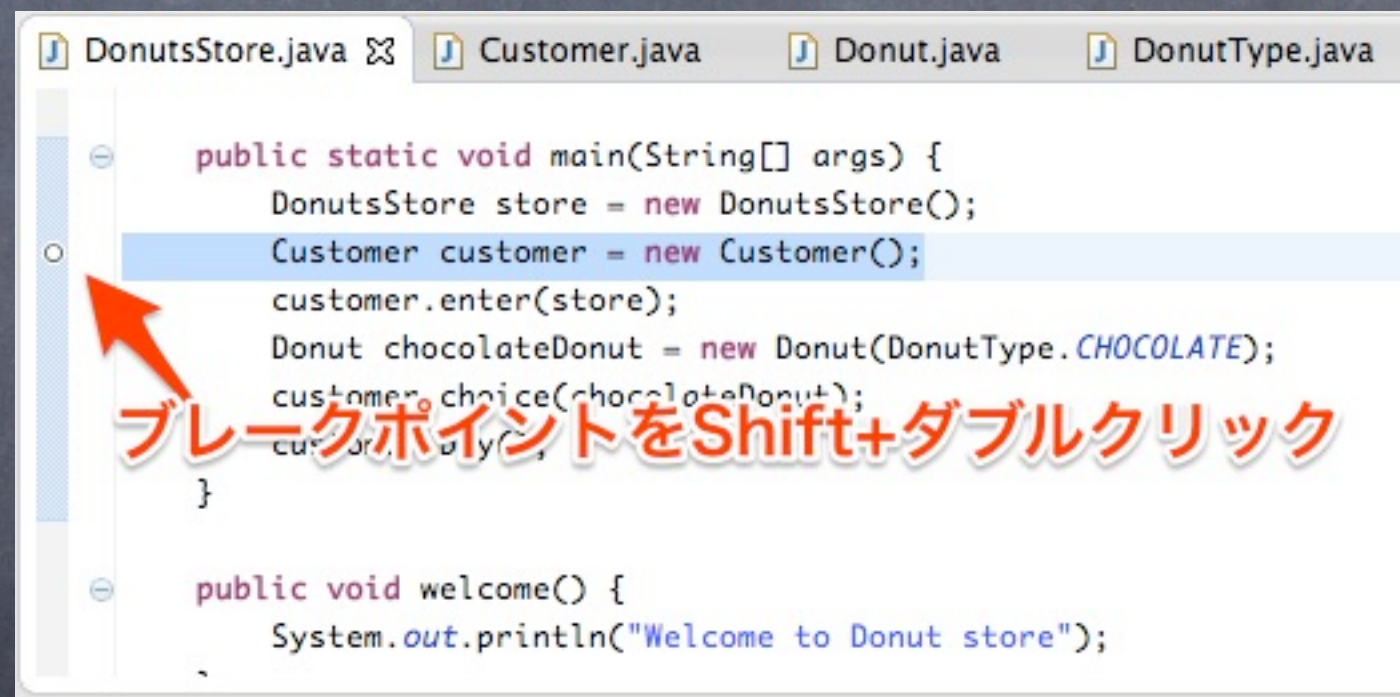
1. Displayビューを表示(Window->Views->Display)
2. コードを入力、選択し、Execute



Q6. 実行中のブレイクポイントを一時的に無効にできますか？

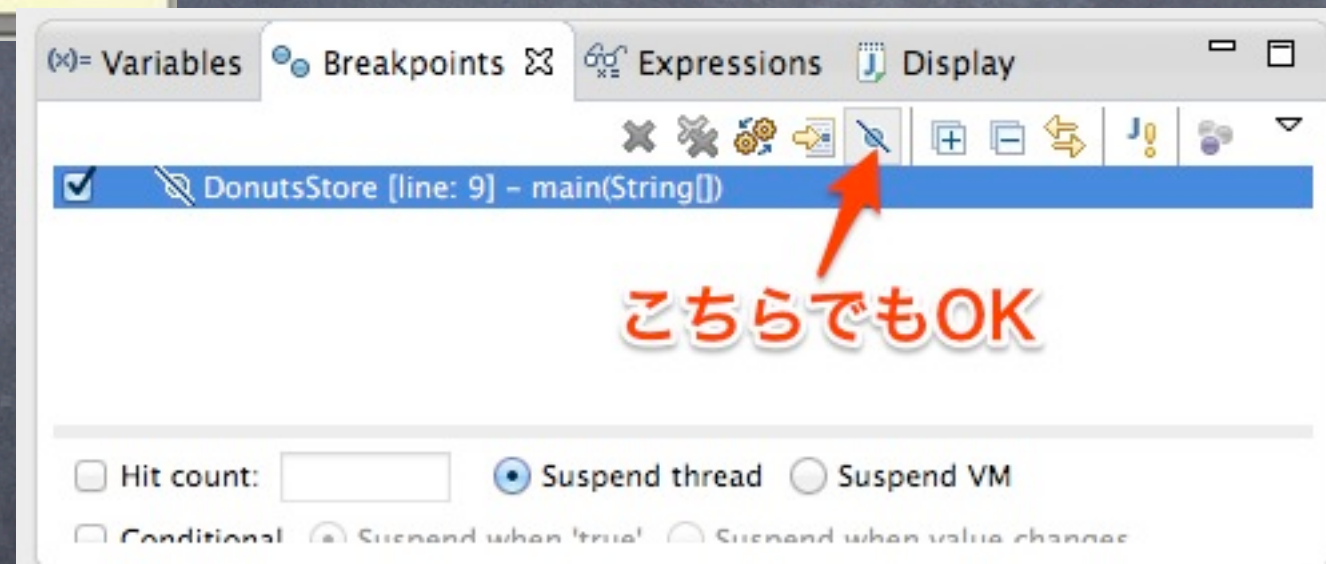
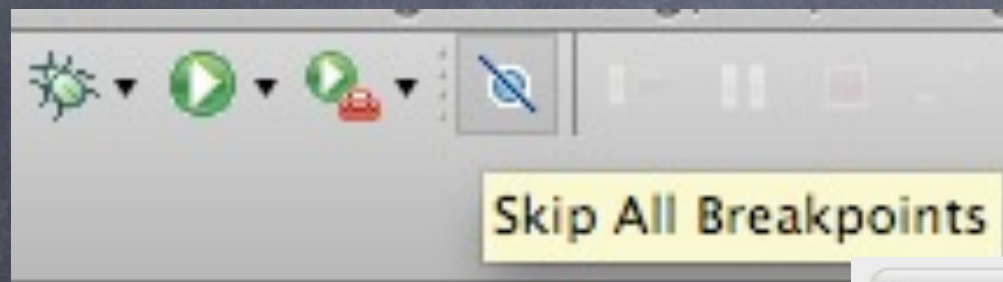
A1. ブレークポイントの無効化

1. Shift++ダブルクリック



A1. ブレークポイントの 全無効化

1. ツールバー上のSkip All Breakpointsをクリック



Q7. 便利なショートカット
を教えてください。

A1. Step Into(F5)

実行中のメソッドの中に入ります。



←のアイコンでもOK

```
DonutsStore.java  Customer.java  Donut.java  DonutType.java

public static void main(String[] args) {
    DonutsStore store = new DonutsStore();
    Customer customer = new Customer();
    customer.enter(store);
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);
    customer.choice(chocolateDonut);
    customer.buy();
}

public void welcome() {
    System.out.println("Welcome to Donut store");
}
```

ここでF5

```
DonutsStore.java  Customer.java  Donut.java  DonutType.java

    this.donuts.add(donut);
}

public void enter(DonutsStore store) {
    this.store = store;
    store.welcome();
}

public void buy() {
    store.order(donuts);
}
}
```

呼び出し先の処理に入ります

A2. Step Over(F6)

次の処理に移ります。



←のアイコンでもOK

```
DonutsStore.java  Customer.java ✕  Donut.java  DonutType.java
    this.donuts.add(donut);
}
public void enter(DonutsStore store) {
    this.store = store;
    store.welcome();
}
public void buy() {
    store.order(donuts);
}
}
```

ここでF6

```
DonutsStore.java  Customer.java ✕  Donut.java  DonutType.java
    this.donuts.add(donut);
}
public void enter(DonutsStore store) {
    this.store = store;
    store.welcome();
}
public void buy() {
    store.order(donuts);
}
}
```

次の処理に移ります

A3. Step Return(F7)

現在のメソッド処理を終えます。



←のアイコンでもOK

```
DonutsStore.java Customer.java Donut.java DonutType.java
    this.donuts.add(donut);
}
public void enter(DonutsStore store) {
    this.store = store;
    store.welcome();
}
public void buy() {
    store.order(donuts);
}
}
```

```
DonutsStore.java Customer.java Donut.java DonutType.java
public static void main(String[] args) {
    DonutsStore store = new DonutsStore();
    Customer customer = new Customer();
    customer.enter(store);
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);
    customer.choice(chocolateDonut);
    customer.buy(chocolateDonut);
}
public void welcome() {
    System.out.println("Welcome to Donut store");
}
```


A4. Resume(F8)

次のブレークポイントまで処理を飛ばします。



←のアイコンでもOK

DonutsStore.java DonutsStore.java DonutsStore.java DonutsStore.java

```
public static void main(String[] args) {  
    DonutsStore store = new DonutsStore();  
    Customer customer = new Customer();  
    customer.enter(store);  
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);  
    customer.choice(chocolateDonut);  
    customer.buy();  
}
```

ここでF8

DonutsStore.java DonutsStore.java DonutsStore.java DonutsStore.java

```
public static void main(String[] args) {  
    DonutsStore store = new DonutsStore();  
    Customer customer = new Customer();  
    customer.enter(store);  
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);  
    customer.choice(chocolateDonut);  
    customer.buy();  
}
```

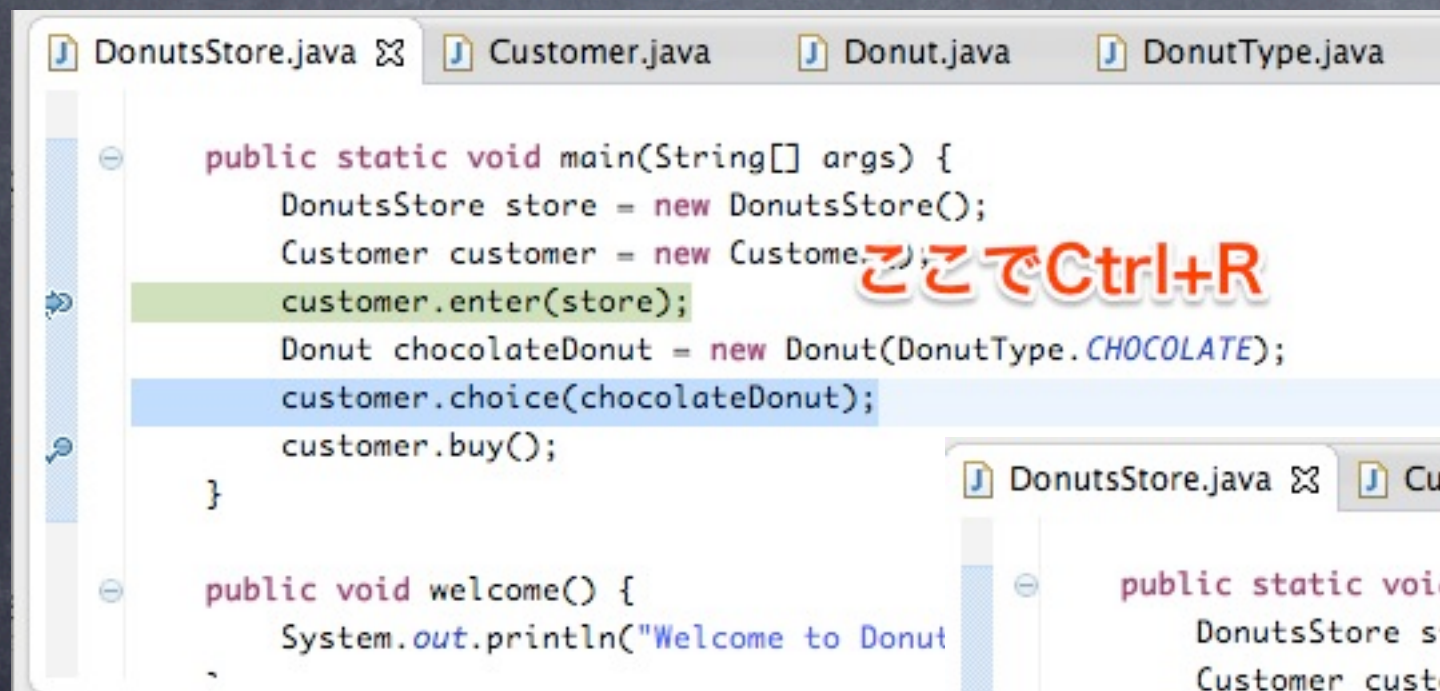
次のブレークポイントへ
ブレークポイントがなければ
デバッグを終了

```
public void welcome() {  
    System.out.println("Welcome to Donut store");  
}
```


A5. Run to Line(Ctrl+R)

カーソルのある行まで処理を飛ばします。

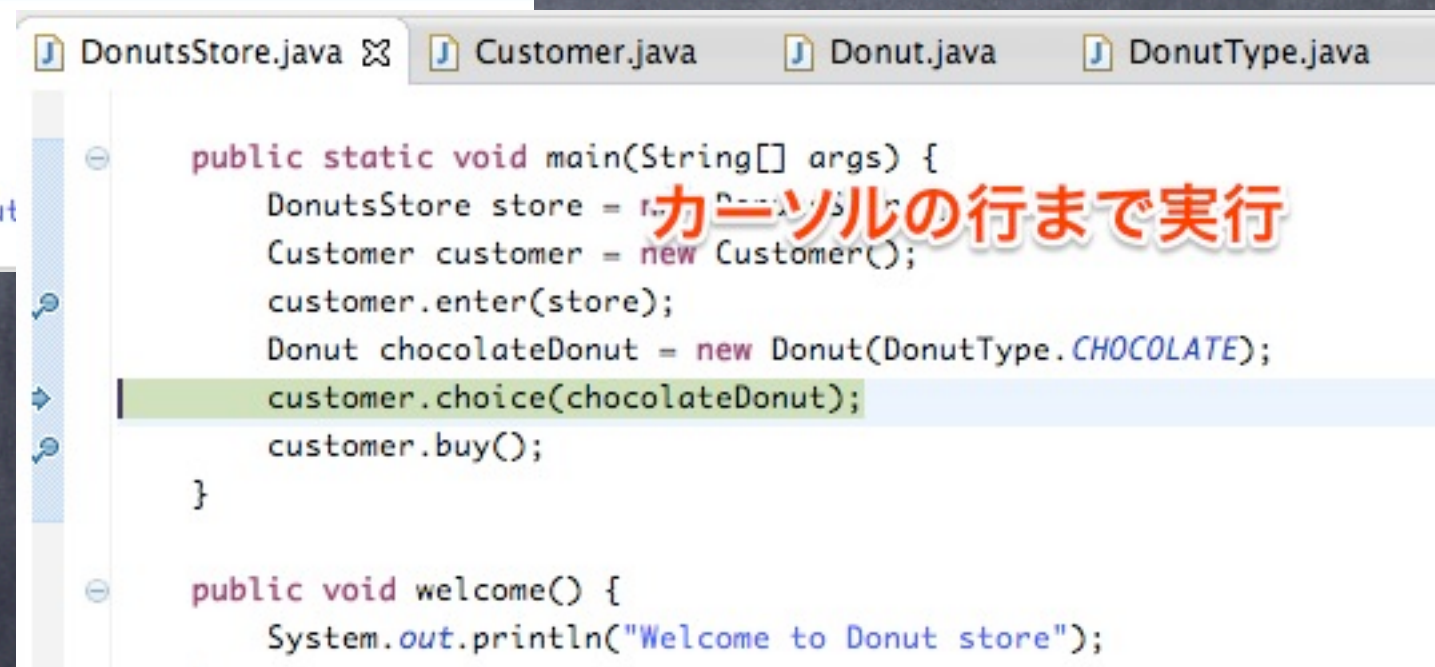
右クリックからも実行可(Mac:Command+R)



```
DonutsStore.java Customer.java Donut.java DonutType.java

public static void main(String[] args) {
    DonutsStore store = new DonutsStore();
    Customer customer = new Customer();
    customer.enter(store);
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);
    customer.choice(chocolateDonut);
    customer.buy();
}

public void welcome() {
    System.out.println("Welcome to Donut");
}
```



```
DonutsStore.java Customer.java Donut.java DonutType.java

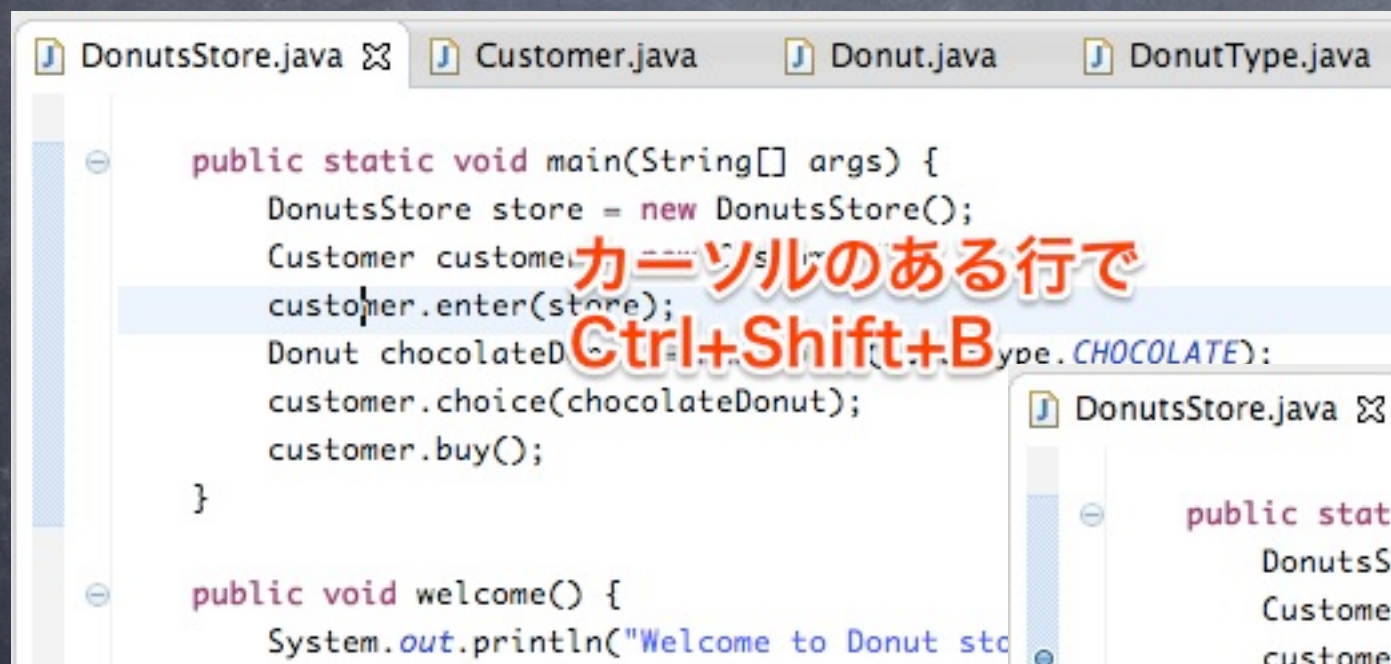
public static void main(String[] args) {
    DonutsStore store = new DonutsStore();
    Customer customer = new Customer();
    customer.enter(store);
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);
    customer.choice(chocolateDonut);
    customer.buy();
}

public void welcome() {
    System.out.println("Welcome to Donut store");
}
```


A6. Toggle Breakpoint(Ctrl+Shift+B)

ブレークポイントを設定/解除します

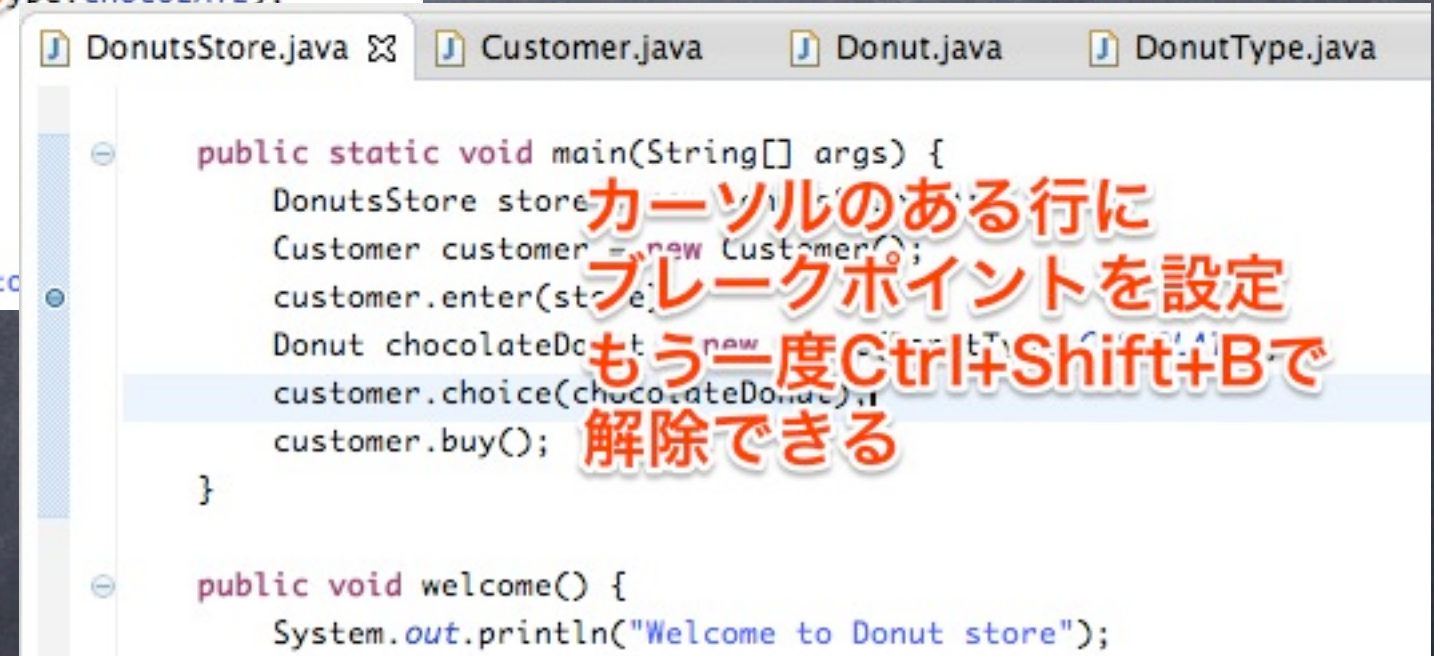
(Mac: Command+Shift+B)



```
DonutsStore.java Customer.java Donut.java DonutType.java

public static void main(String[] args) {
    DonutsStore store = new DonutsStore();
    Customer customer = new Customer();
    customer.enter(store);
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);
    customer.choice(chocolateDonut);
    customer.buy();
}

public void welcome() {
    System.out.println("Welcome to Donut store");
}
```



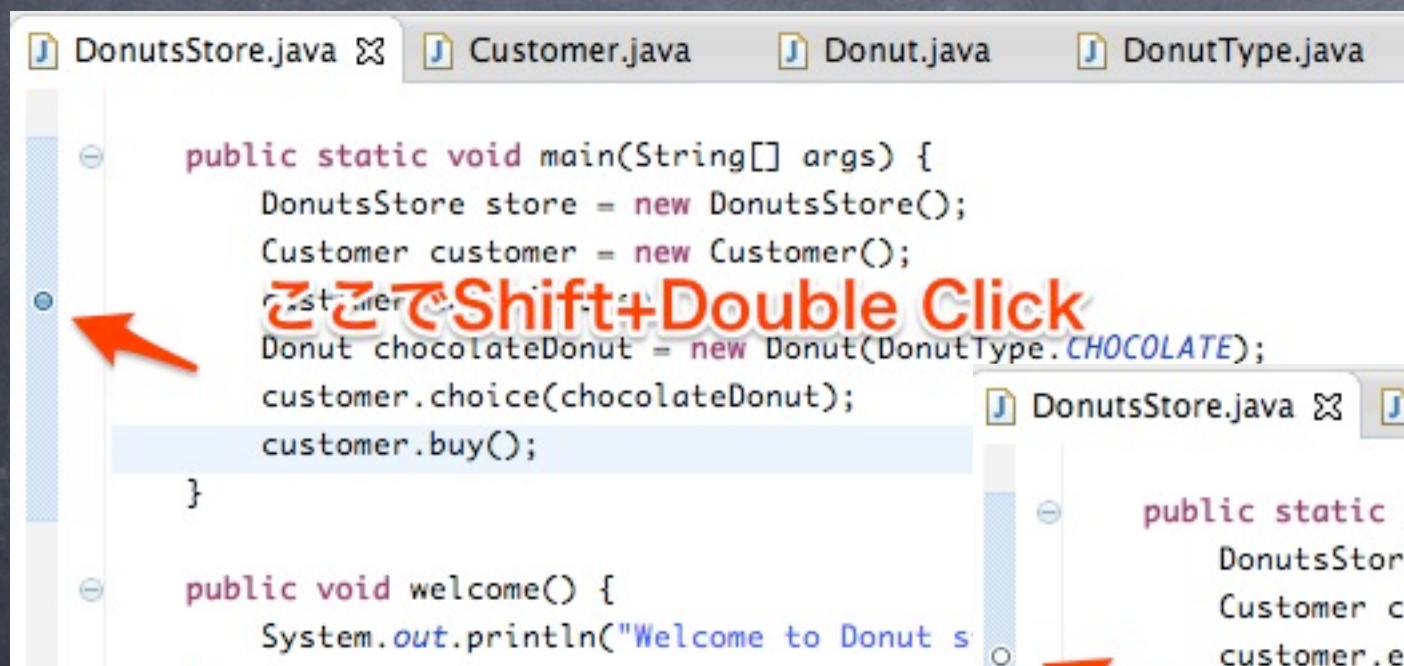
```
DonutsStore.java Customer.java Donut.java DonutType.java

public static void main(String[] args) {
    DonutsStore store = new DonutsStore();
    Customer customer = new Customer();
    customer.enter(store);
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);
    customer.choice(chocolateDonut);
    customer.buy();
}

public void welcome() {
    System.out.println("Welcome to Donut store");
}
```


A7. Disable Breakpoint (Shift+Double Click)

ブレークポイントを無効にします



```
DonutsStore.java Customer.java Donut.java DonutType.java

public static void main(String[] args) {
    DonutsStore store = new DonutsStore();
    Customer customer = new Customer();
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);
    customer.choice(chocolateDonut);
    customer.buy();
}

public void welcome() {
    System.out.println("Welcome to Donut s");
}
```



```
DonutsStore.java Customer.java Donut.java DonutType.java

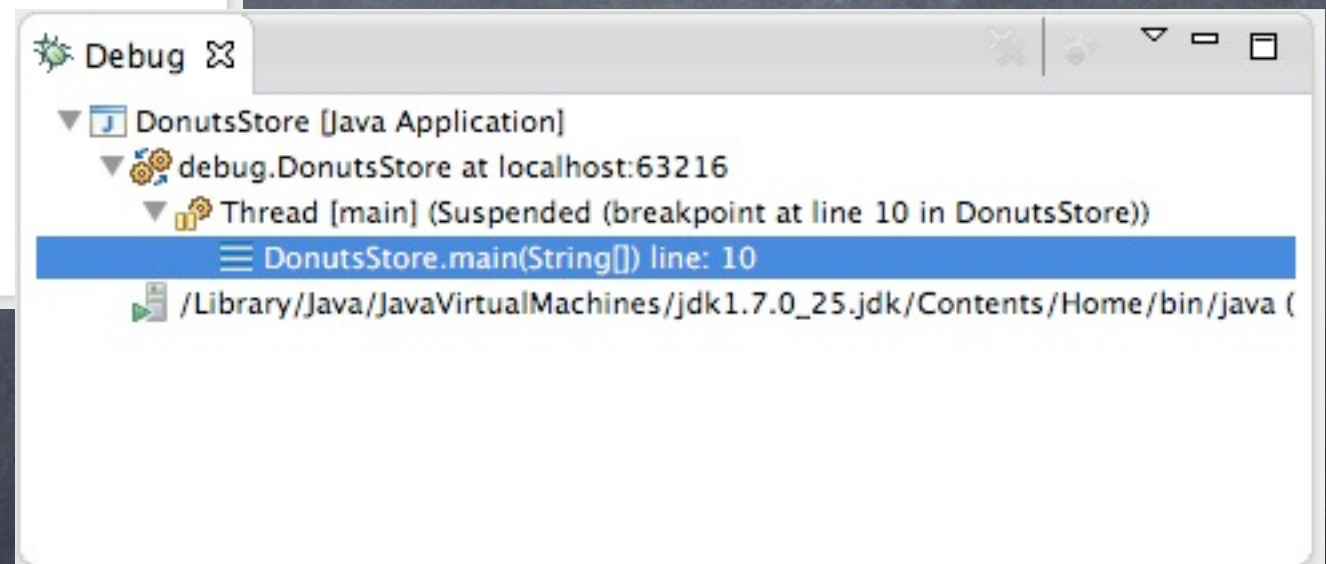
public static void main(String[] args) {
    DonutsStore store = new DonutsStore();
    Customer customer = new Customer();
    customer.enter(store);
    Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);
    customer.choice(chocolateDonut);
    customer.buy();
}

public void welcome() {
    System.out.println("Welcome to Donut store");
}
```

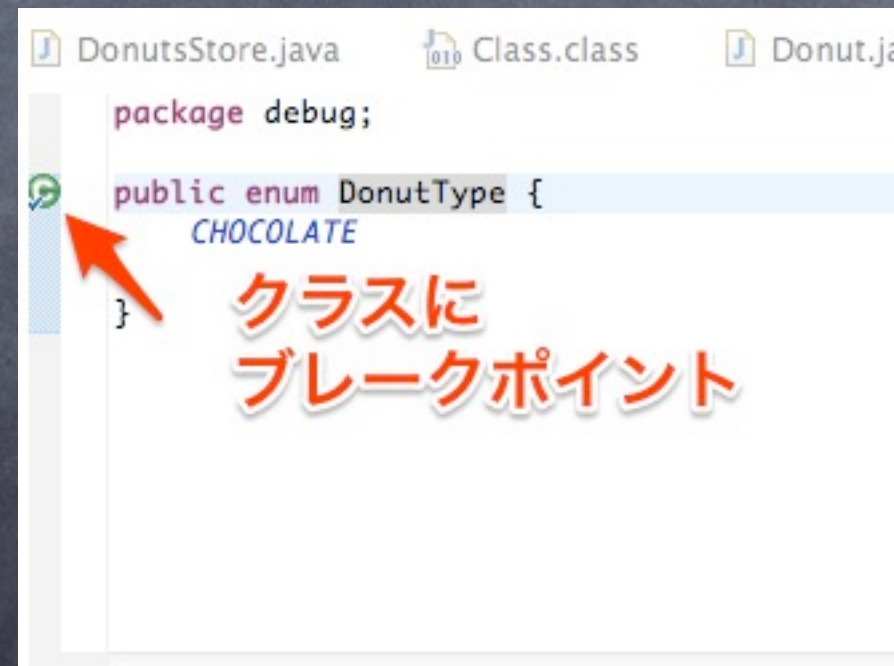

A8. Debug(F11)

前回実行したデバッグを再実行します。

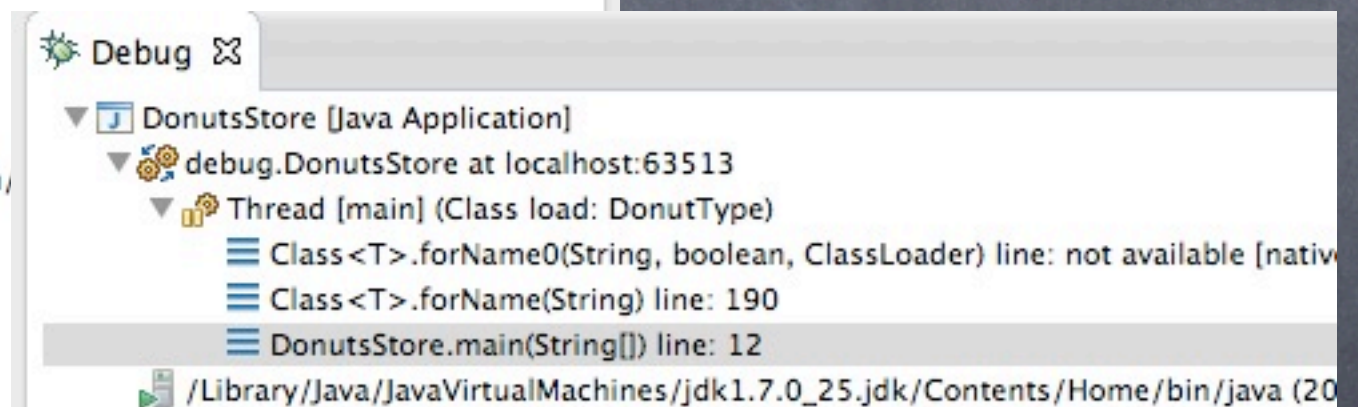
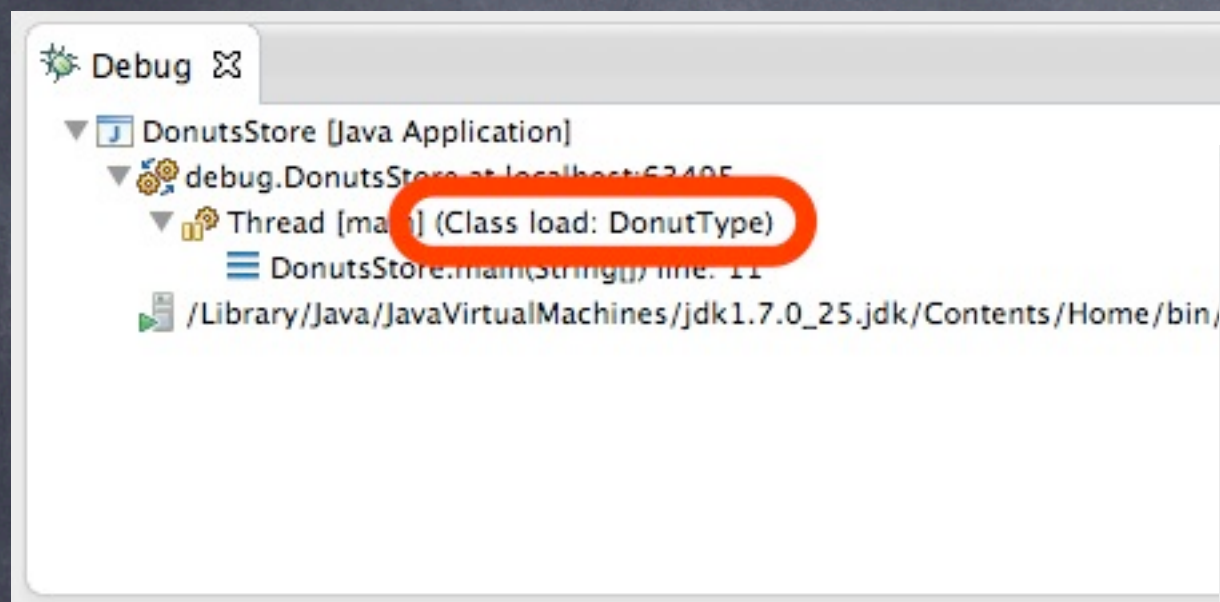
(Mac:Command+F11)



Q8. クラスにブレークポイントを貼るとどうなりますか？



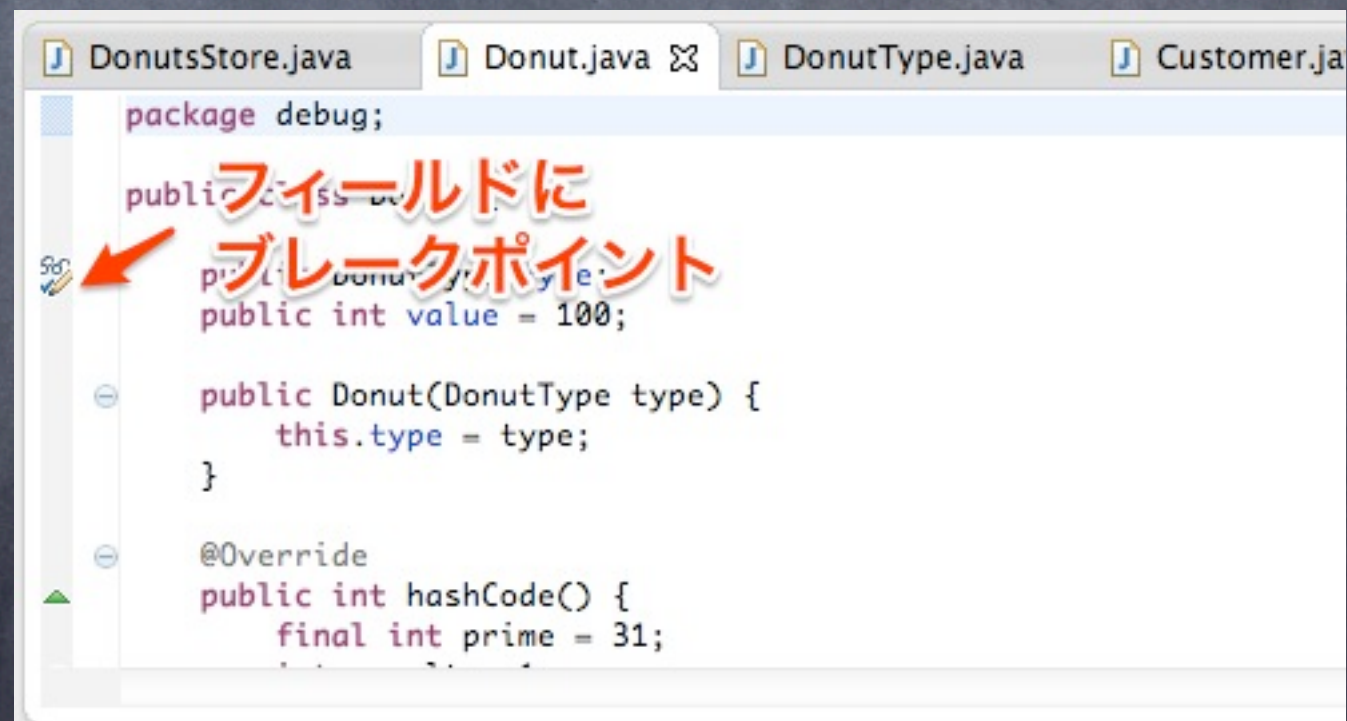
A. クラスロード時に停止



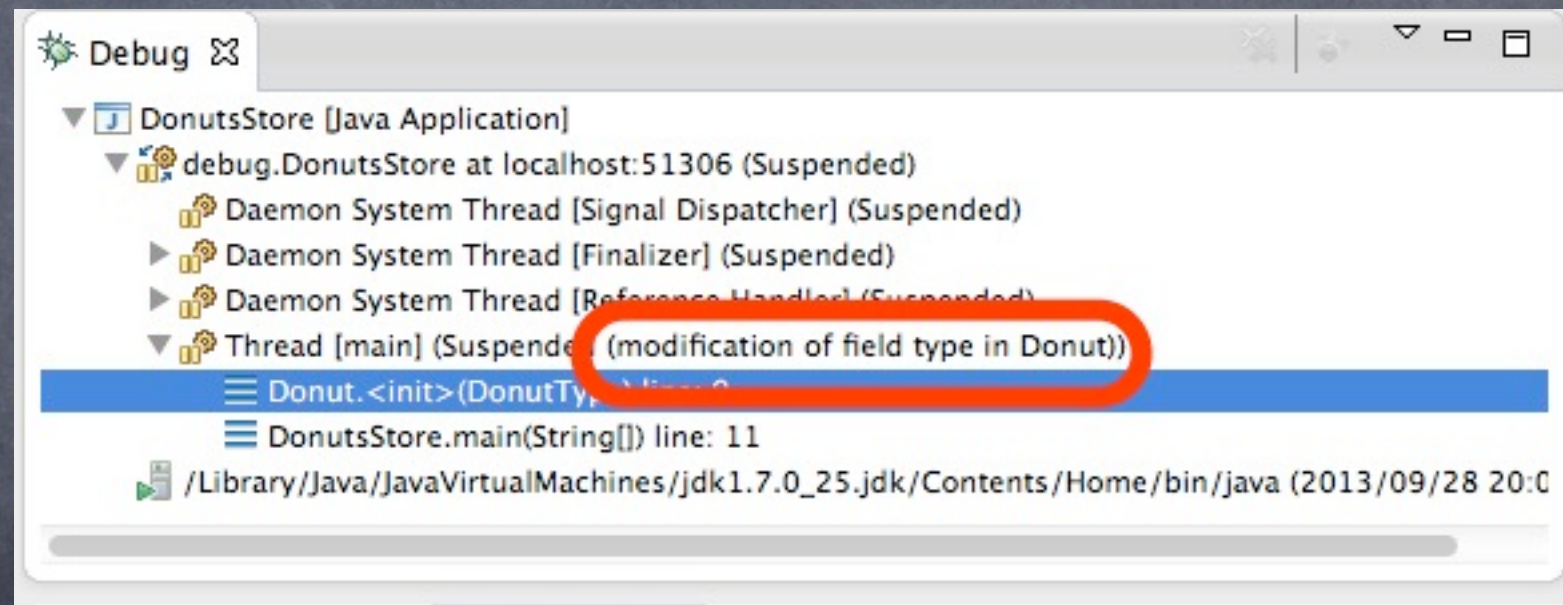
リフレクション
呼び出しも対応



Q9. フィールドにブレーク
ポイントを貼るとどうなりま
すか？



A. フィールドにアクセス/ 更新があった時に停止



Q10. フィールドを更新した時にのみ停止するようにするには、どうしたらいいですか？



The screenshot shows an IDE with four tabs: DonutsStore.java, Donut.java, DonutType.java, and Customer.java. The Donut.java tab is active, displaying the following code:

```
package debug;

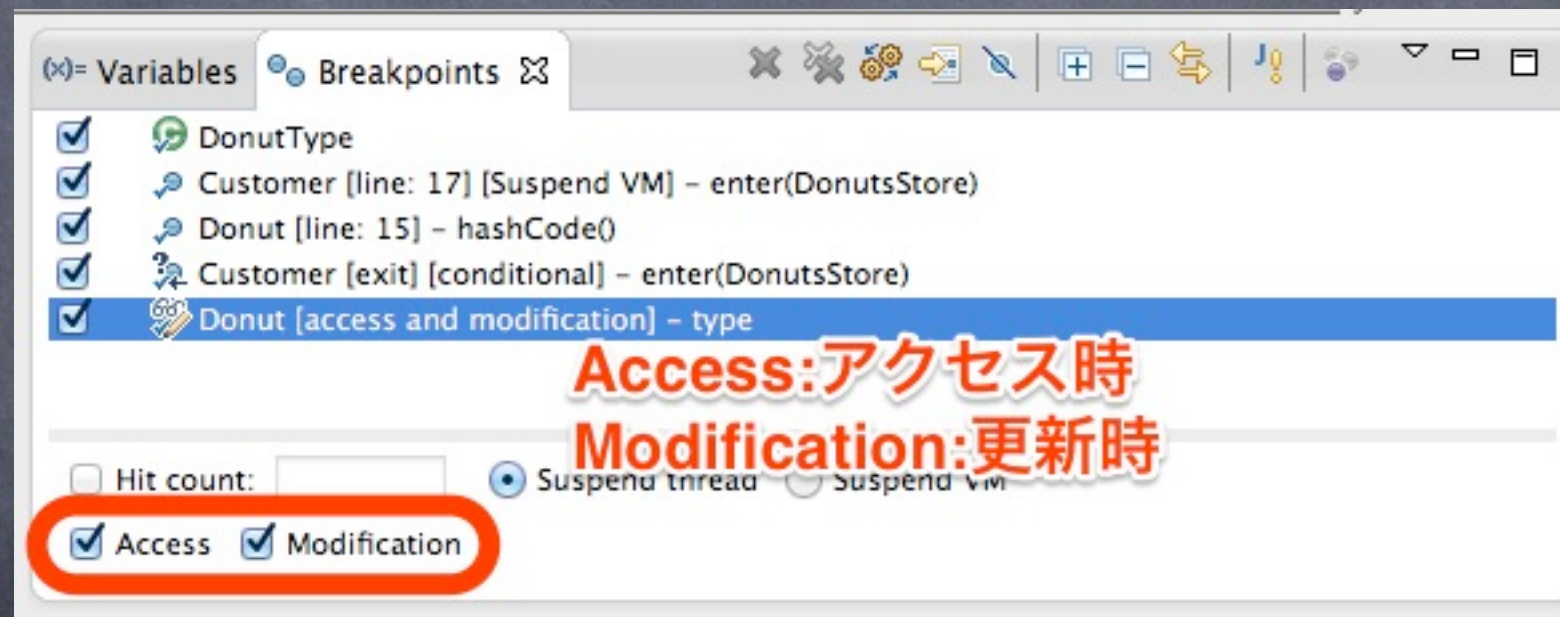
public class Donut {
    public int value = 100;

    public Donut(DonutType type) {
        this.type = type;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
    }
}
```

Red Japanese text is overlaid on the code: "フィールドに" (Field to) and "ブレークポイント" (Breakpoint). A red arrow points to the line `public int value = 100;`, indicating where a breakpoint has been set.

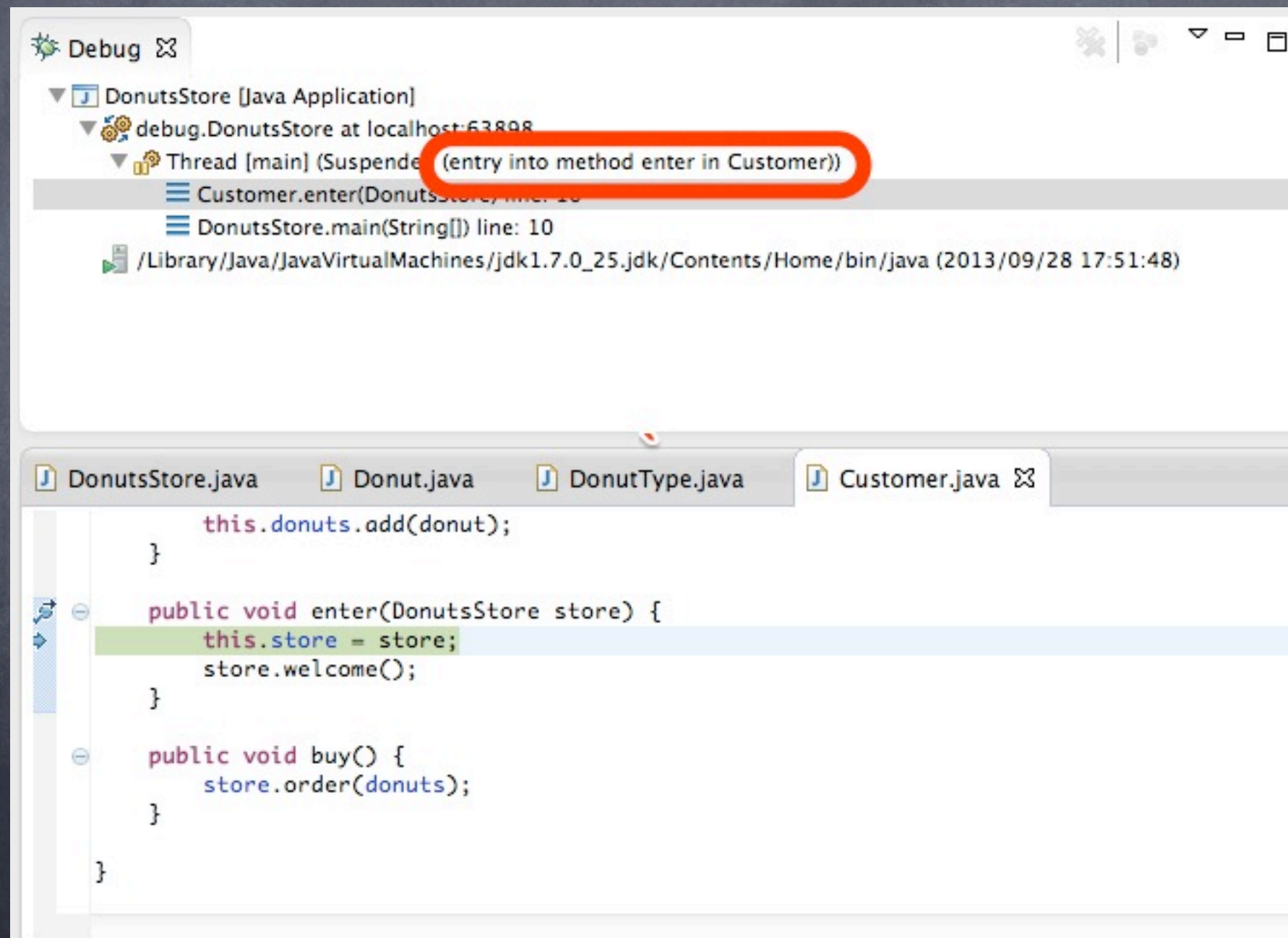
A. ブレークポイントのプロパティで調整



Q11. メソッドにブレークポイントを貼るとどうなりますか？



A. メソッド呼び出し時に 停止



Q12. mainが呼び出されるタイミングで停止させるにはどうしたらいいですか？

A1. mainメソッドにブレーク ポイントを貼る

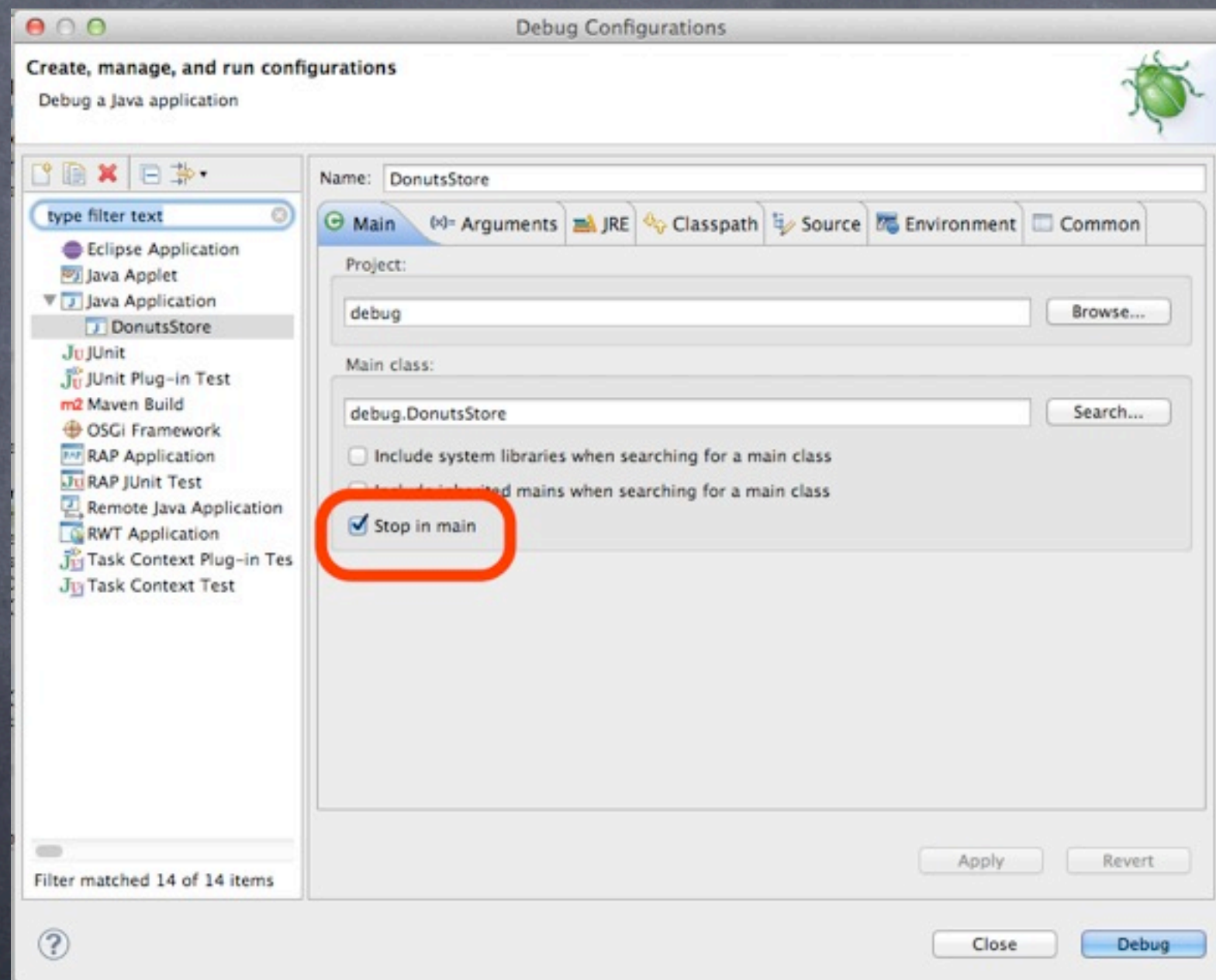


The screenshot shows an IDE with four tabs: DonutsStore.java, Donut.java, DonutType.java, and Customer.java. The DonutsStore.java file is open, showing the following code:

```
public class DonutsStore {  
    public static void main(String[] args) {  
        DonutsStore store = new DonutsStore();  
        Customer customer = new Customer();  
        customer.enter(store);  
        Donut chocolateDonut = new Donut(DonutType.CHOCOLATE);  
        customer.choice(chocolateDonut);  
        customer.buy();  
    }  
  
    public void welcome() {  
        System.out.println("Welcome to Donut store");  
    }  
}
```

A red arrow points to the first line of the `main` method, `DonutsStore store = new DonutsStore();`, where a breakpoint has been set. The breakpoint icon is a small circle with a red dot inside, located on the left margin of the code editor.

A2. デバグのランチャー に指定する



Q13. methodがreturnされる
タイミングで停止できま
せんか？

A. メソッドのブレークポイントのプロパティで指定できます。

The screenshot displays the 'Properties for debug.Customer - enter(DonutsStore)' dialog box. The 'Method Breakpoint' tab is active. The 'Type' is set to 'debug.Custor' and the 'Method' is 'enter(DonutsStore)'. The 'Exit' checkbox is checked, and the 'Entry' checkbox is unchecked. A red circle highlights the 'Exit' checkbox. A red arrow points to the 'Exit' checkbox. The text 'Exitにチェックを入れるとメソッドから抜ける時に停止' is overlaid on the dialog. The 'DonutsStore.java' file is open in the background, showing the 'enter' method. A red arrow points to the 'enter' method, and the text 'アイコンも変化;' is overlaid on the code.

Properties for debug.Customer - enter(DonutsStore)

Method Breakpoint

Type: debug.Custor
Method: enter(DonutsStore)

Exitにチェックを入れると
メソッドから抜ける時に停止

☒ Enabled
☐ Hit count:
☒ Suspend thread ☐ Suspend VM ☐ Entry ☒ Exit
☐ Conditional ☒ Suspend when 'true' ☐ Suspend when value changes
<Choose a previously entered condition>

DonutsStore.java Donut.java DonutType.java Customer.java

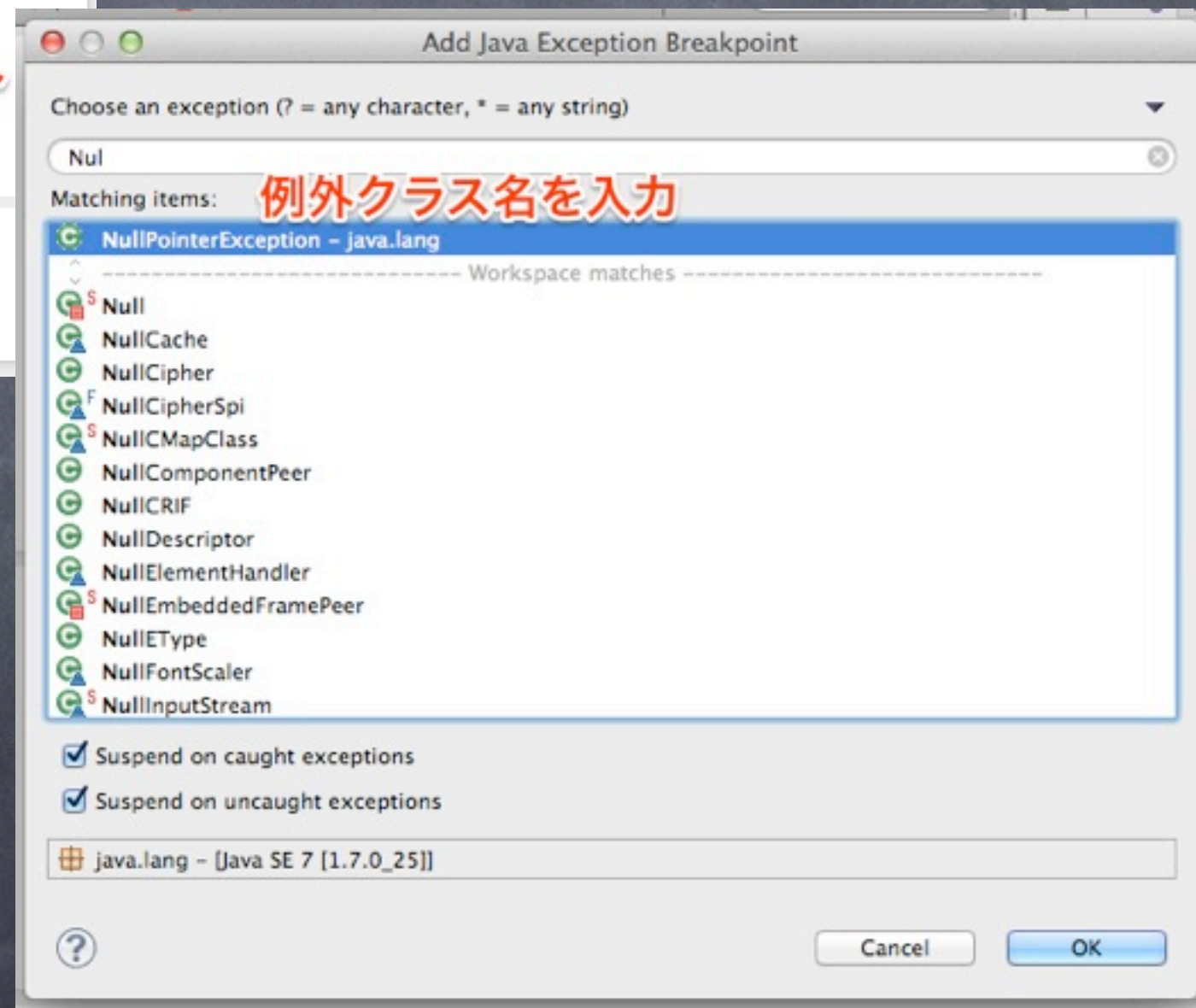
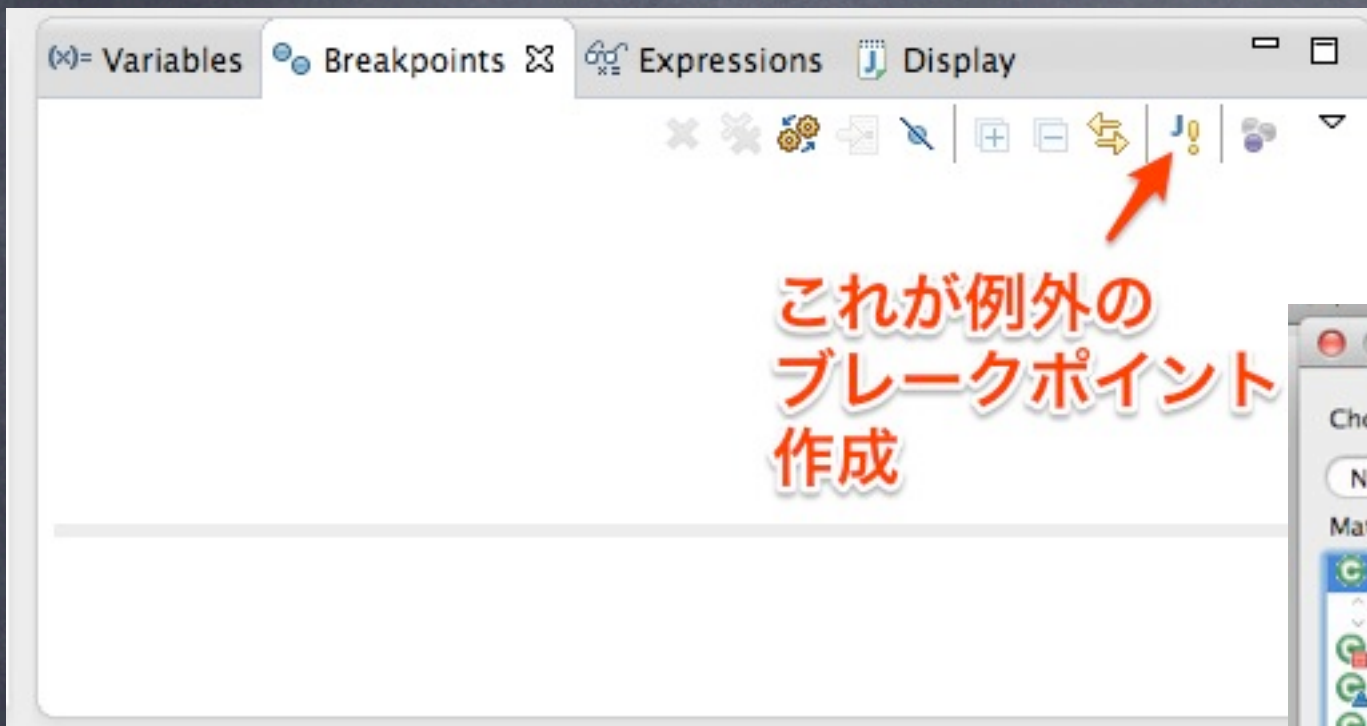
アイコンも変化;

```
public void enter(DonutsStore store) {  
    this.store = store;  
    if(store == null){  
        System.out.println("store is null");  
    } return;  
    } else {  
        store.welcome();  
    }  
}
```

```
public void buy() {
```


Q14. 例外が発生したタイミングで停止できませんか？

A. ブレークポイントビュー
から指定できます。



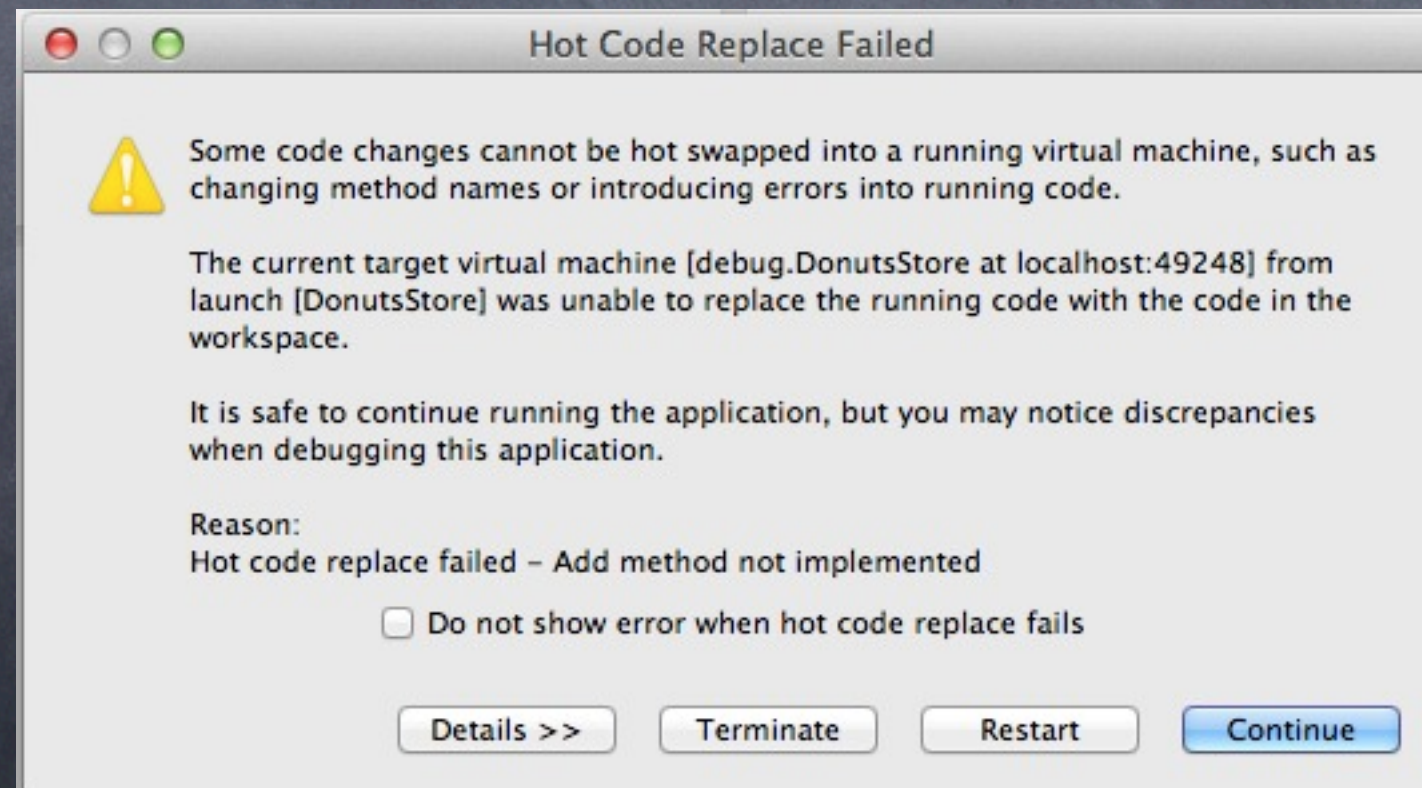
応用編

Q15. 実行中にコードを書き換えると、自動で処理を置き換えられますか？

A. デバッグ中はホットコードリプレイスが有効

ホットコードリプレイスとは、実行中のコードを書き換えられる機構です。

※メソッド等のシグネチャ(クラスやメソッドの名前やパラメータ名、型の指定)が変わると、リプレイスが無効になり、下記のダイアログを表示。



Q16. 実環境でデバッグで
きませんか？

A. リモートデバッグしましょう

- リモートデバッグとは、他のJVMをデバッグする事です。
- JDWP(Java Debug Wire Protocol)を使って通信できるように起動時に指定します。
- 他のマシンのJVMとも通信できます。
- 知っておくと結構べんり。(ant等のデバッグ)
- 動作しているJVMのクラスを書き換えないとホットコードリプレイスはされない点は注意

A. リモートデバッグしまし よう

● アプリ側の引数例

```
-agentlib:jdwp=transport=dt_socket,suspend=y,address=localhost:44000
```


A. リモートデバッグしよう

- それぞれの意味

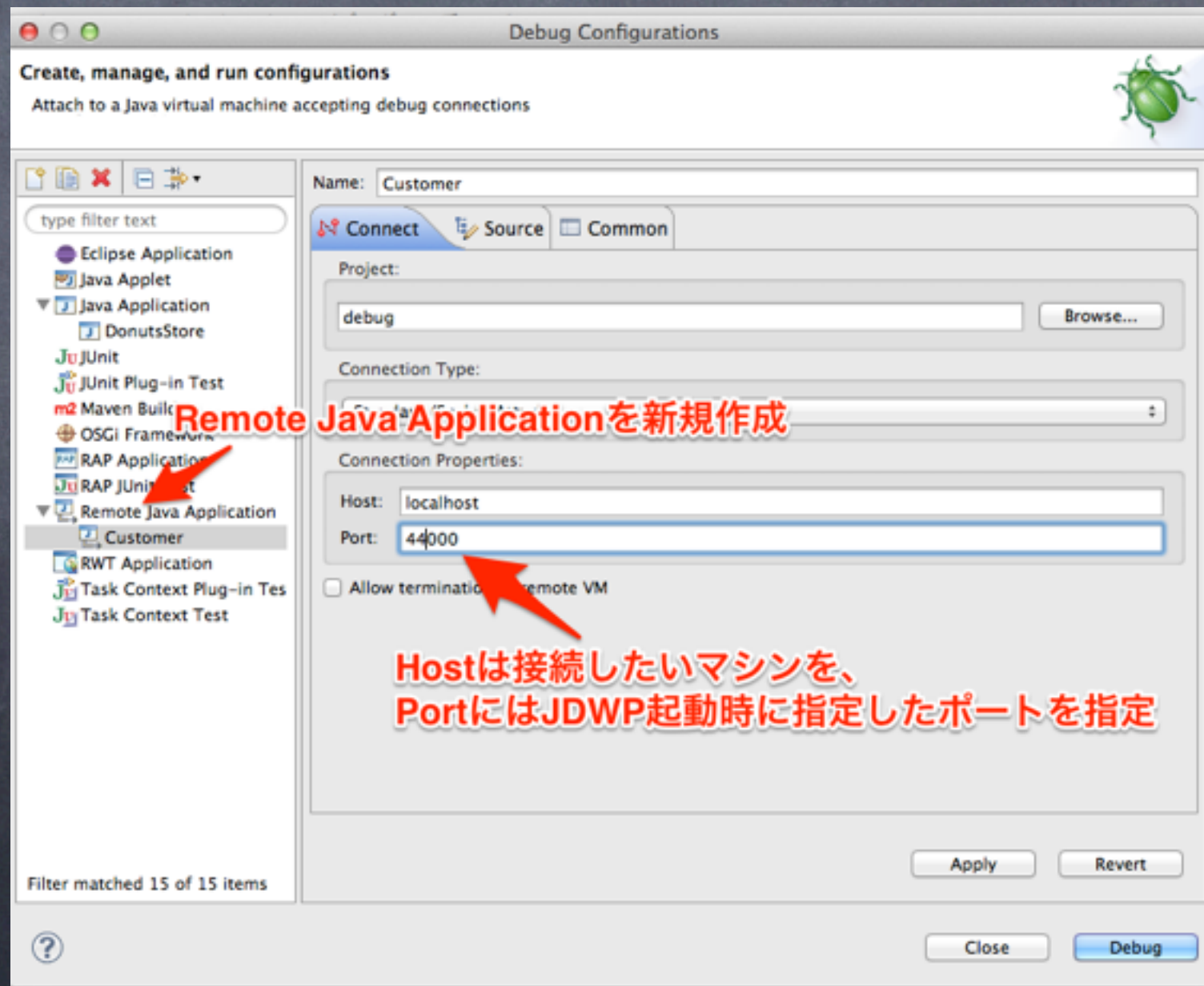
```
-agentlib:jdwp=transport=dt_socket,suspend=y,address=localhost=44000
```

- 「-agentlib:jdwp」 JDWP実装のロード指定。
- 「transport」 通信方法。通常はソケット。Windowsでは共有メモリも指定可。
- 「server」 yの場合、起動するJVMがデバッグされる側になります。デバッグしたいアプリ側はy
- 「suspend」 yの場合、接続するまでアプリは起動されません。
- 「address」 接続のためのアドレスです。アプリ側はポートを指定

A. リモートデバッグしまし よう

• デバッガ側

- Debug Configurations... を開く(虫アイコンの右の下三角から)



Q17. 実環境でmainが実行された直後からデバッグする方法はありますか？

A. アプリ起動時の指定を「suspend=y」にしましょう

- それぞれの意味

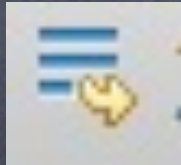
```
-agentlib:jdwp=transport=dt_socket,suspend=y,address=localhost=44000
```

- 「-agentlib:jdwp」 JDWP実装のロード指定。
- 「+transport」 通信方法。通常はソケット。Windowsでは共有メモリも指定可。
- 「server」 yの場合、起動するJVMがデバッグされる側になります。デバッグしたいアプリ側はy
- 「suspend」 yの場合、接続するまでアプリは起動されません。
- 「address」 接続のためのアドレスです。アプリ側はポートを指定

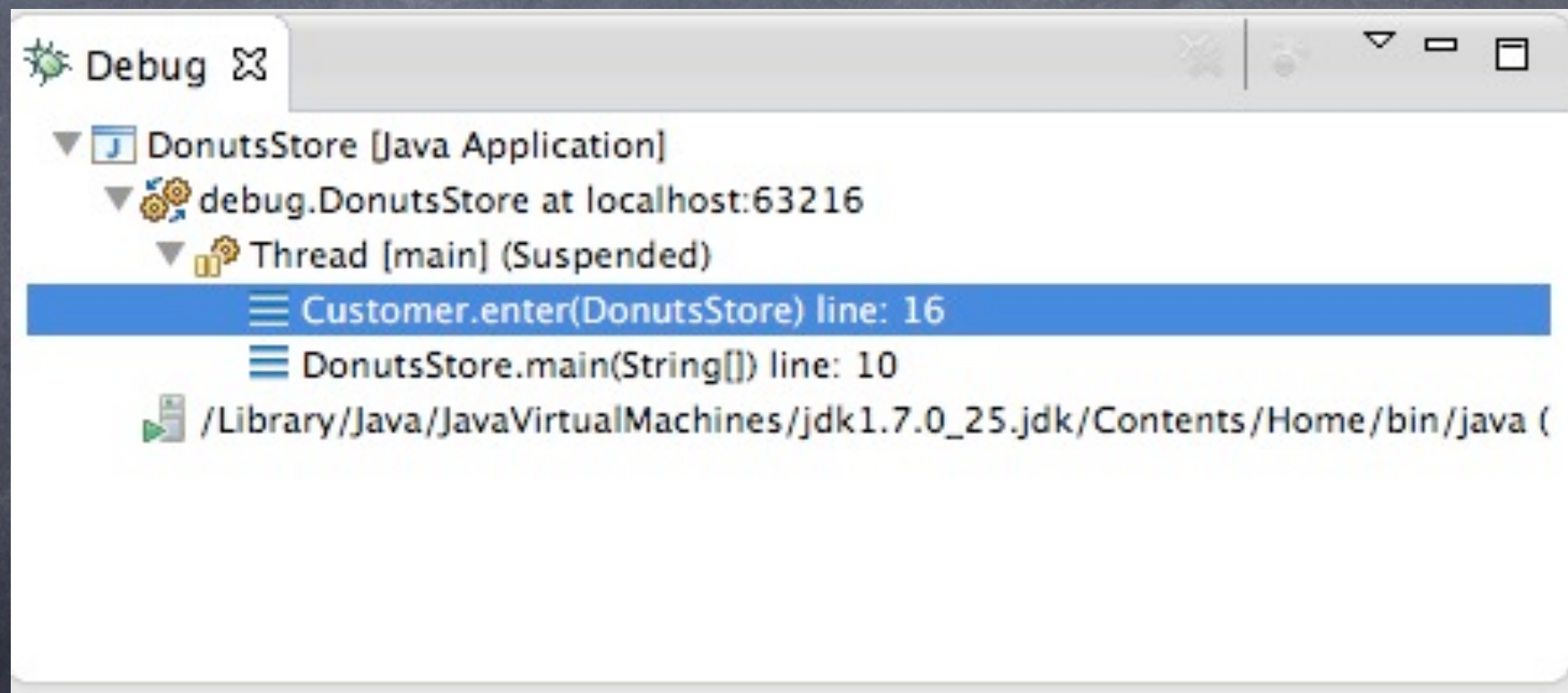
Q18. 間違えてStep Overし
過ぎました。もう一度デバ
ッグをやり直すしかありま
せんか？

A. Drop to Frameする

呼び出しスタック上のFrameを捨てると、そのメソッドが呼び出される前に戻ります。

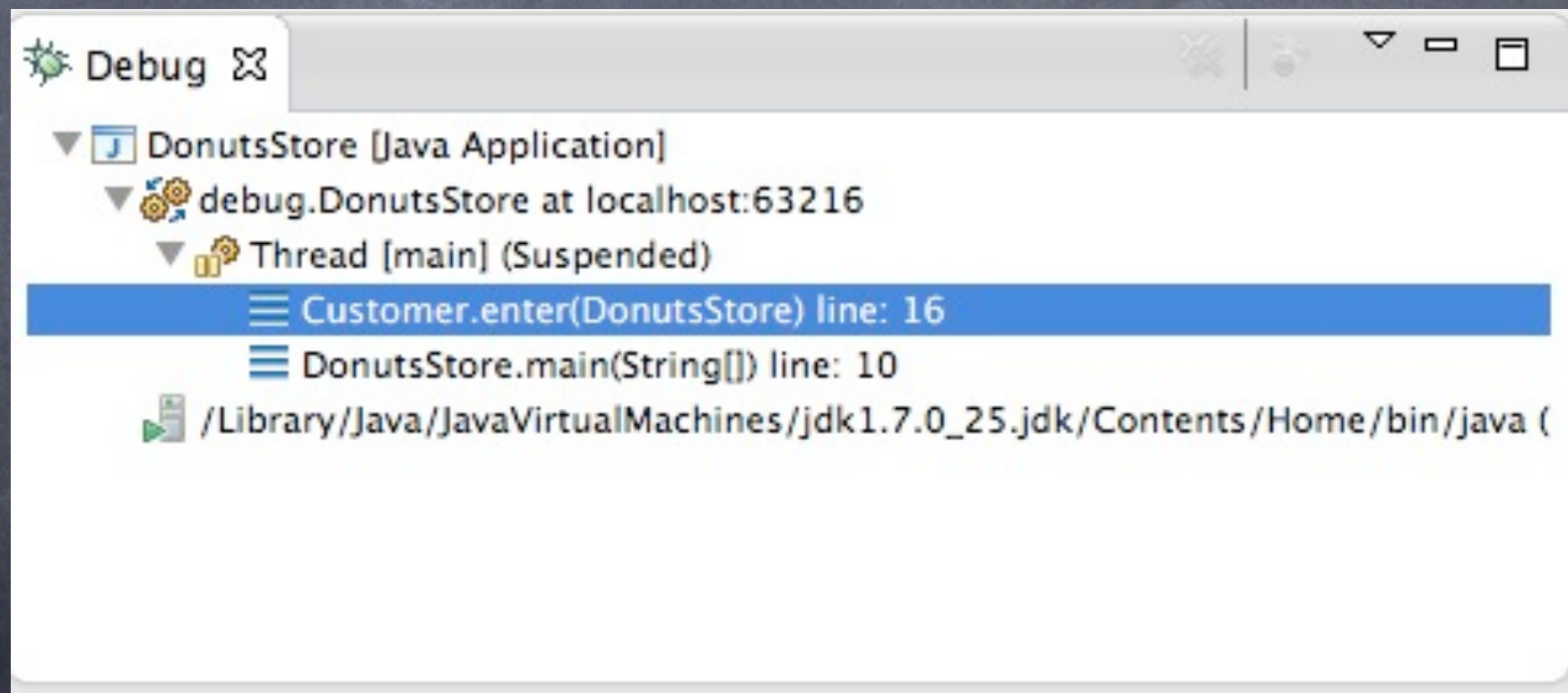


←このアイコンをクリック



A. Drop to Frameする

注意:DBなどの外部リソースのデータや、staticなフィールド等は元に戻りません。



Q19. 呼び出し元のメソッド
の状態を知ることとはできま
せんか？

A. Frameを選びましょう。

呼び出しスタック上のFrameを選択すると、Variablesビュー等選択したフレームの中のものになります。

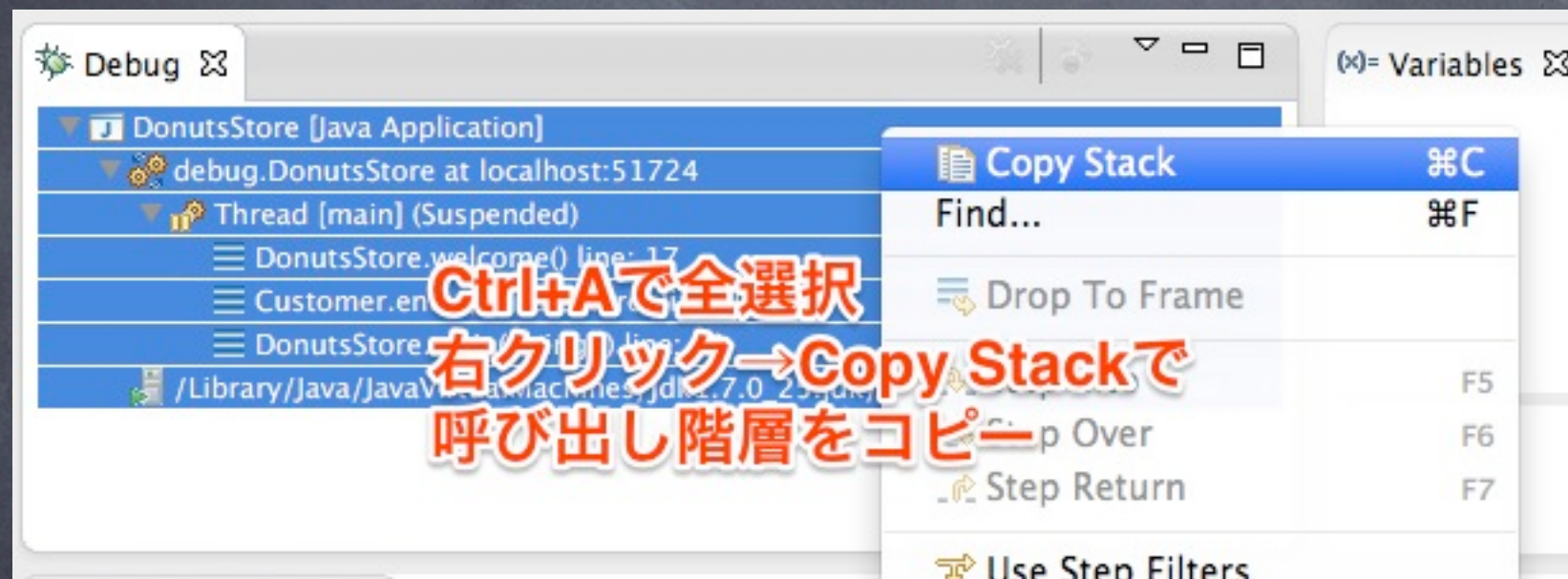
The screenshot displays the IDE's debug interface. The 'Debug' window on the left shows the call stack for 'DonutsStore [Java Application]'. The selected frame is 'Customer.enter(DonutsStore) line: 17'. The 'Variables' window on the right shows the state of the selected frame, with 'this' pointing to a 'Customer' object (id=16) and 'store' pointing to a 'DonutsStore' object (id=27). The 'DonutsStore.java' file is open in the editor, showing the 'enter' method.

```
private DonutsStore store;  
private List<Donut> donuts = new ArrayList<>();  
  
public void choice(Donut donut) {  
    this.donuts.add(donut);  
}  
  
public void enter(DonutsStore store) {  
    this.store = store;  
    store.welcome();  
}
```


Q20. 呼び出し階層をブログに貼りたいです。どうすればできますか？

Thanks @n3104

A. デバッグビューは コピー可能



貼り付け例

DonutsStore [Java Application]

debug.DonutsStore at localhost:51724

Thread [main] (Suspended)

DonutsStore.welcome() line: 17

Customer.enter(DonutsStore) line: 17

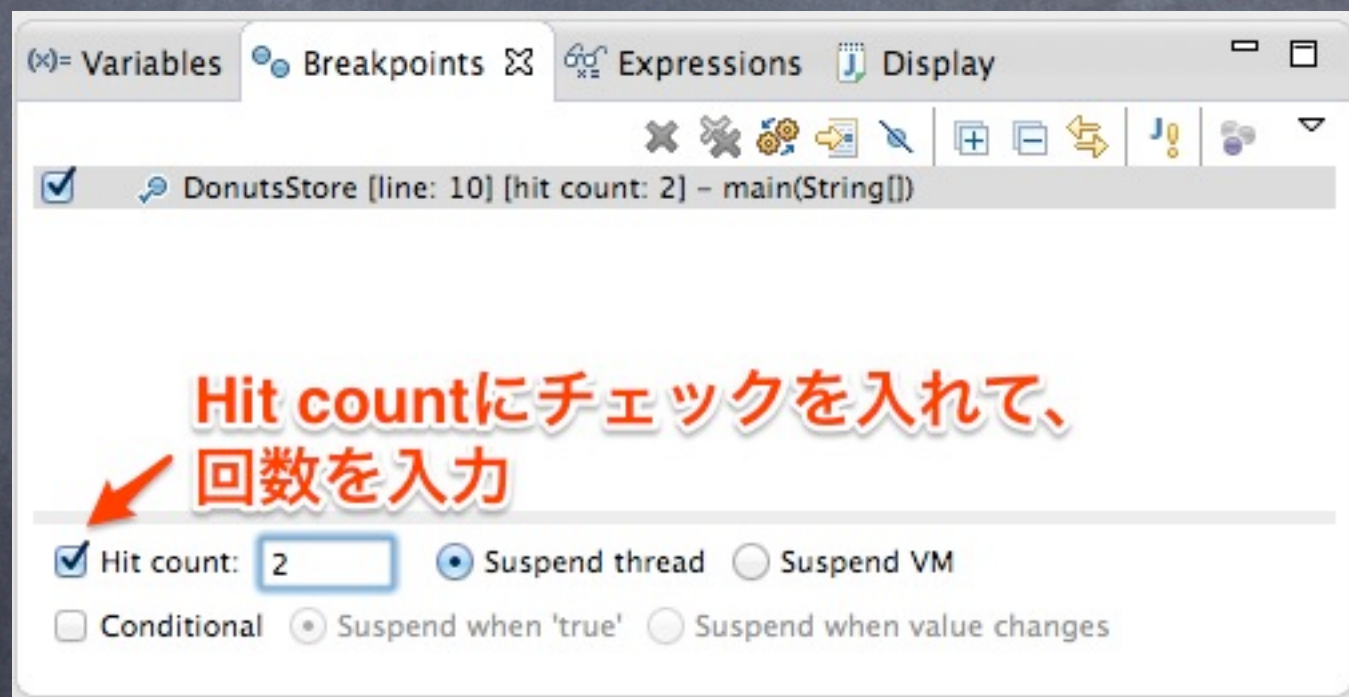
DonutsStore.main(String[]) line: 10

/Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/Contents/Home/bin/java (2013/09/25 21:46:02)

Q21. ブレークポイントに指定した回数処理が通った時に止まるように、ブレークポイントを貼るにはどうすればよいですか？

A. BreakpointにHit Count を指定しましょう。

ブレークポイントに止まる回数を数えてデバッグする
ような状況に重宝します。



Q22. ある条件を満たす
時、初めて止まるようにブ
レークポイントを貼るには
どうすればよいですか？

A. Breakpointのプロパティから条件を指定しましょう。

実行した結果、trueを返した時、停止します。

ブレークポイントを
選択し右クリック
Breakpoint Properties...
を選ぶ

Conditionalにチェック

**Suspend when 'true'を選び、
条件式を入力しましょう。
最終行の値がbooleanで
なければなりません。**

Q23. ある変数が変更された時に停止するには、どう
いう条件を設定すればよ
いですか？

A. Breakpointのプロパティから条件を指定しましょう。

実行した結果、指定した変数が変わった時、停止します。

The image shows a sequence of steps to configure a breakpoint in an IDE. On the left, the 'Breakpoints' window lists several breakpoints, including 'Customer [line: 17]'. A red arrow points to this entry, with the text 'ブレークポイントを選択し右クリック' (Select breakpoint and right-click). A context menu is open over this entry, with 'Breakpoint Properties...' highlighted. A red arrow points to this option, with the text 'Breakpoint Properties...を選ぶ' (Choose Breakpoint Properties...). On the right, the 'Properties for debug.Customer [line:17] - enter(DonutsStore)' dialog is shown. The 'Conditional' checkbox is checked, with a red arrow pointing to it and the text 'Conditionalにチェック' (Check Conditional). Below this, the 'Suspend when value changes' radio button is selected, with a red arrow pointing to it and the text 'Suspend when value changesにチェックを入れ、変更を監視したい変数を入力しましょう。(nullからの変更も監視されます。)' (Check Suspend when value changes, enter the variable you want to monitor for changes. (Changes from null will also be monitored.)).

ブレークポイントを選択し右クリック

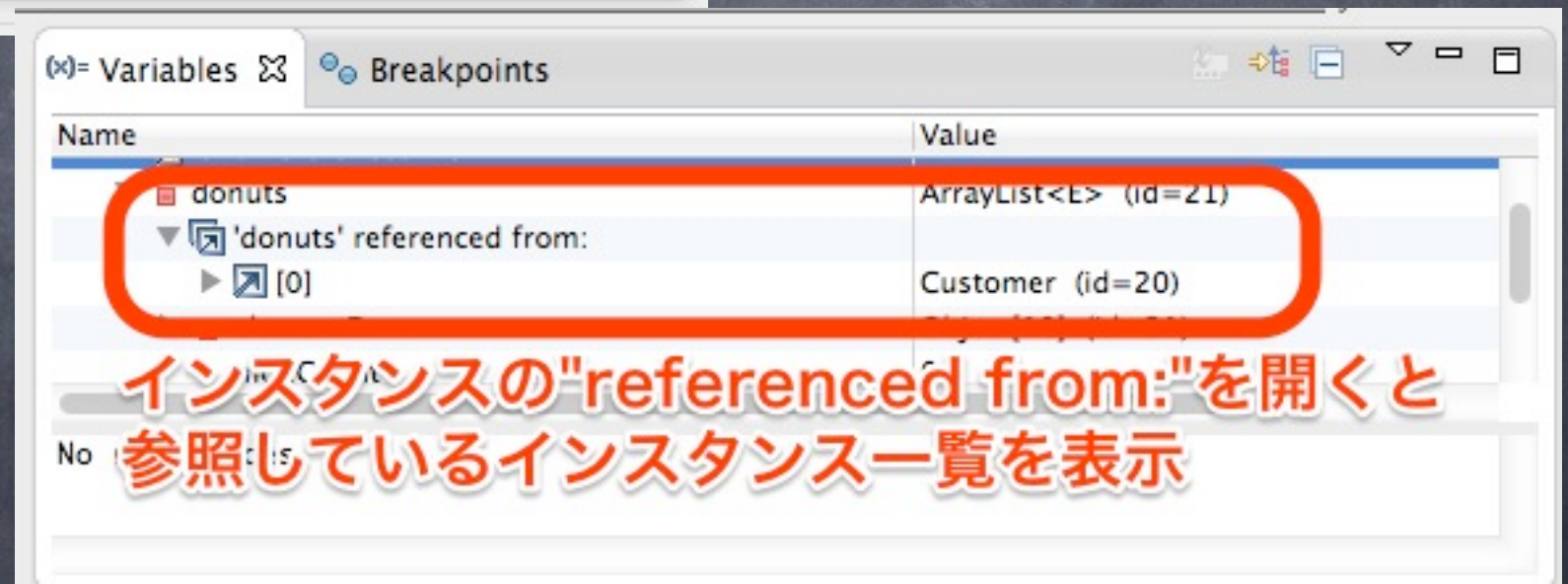
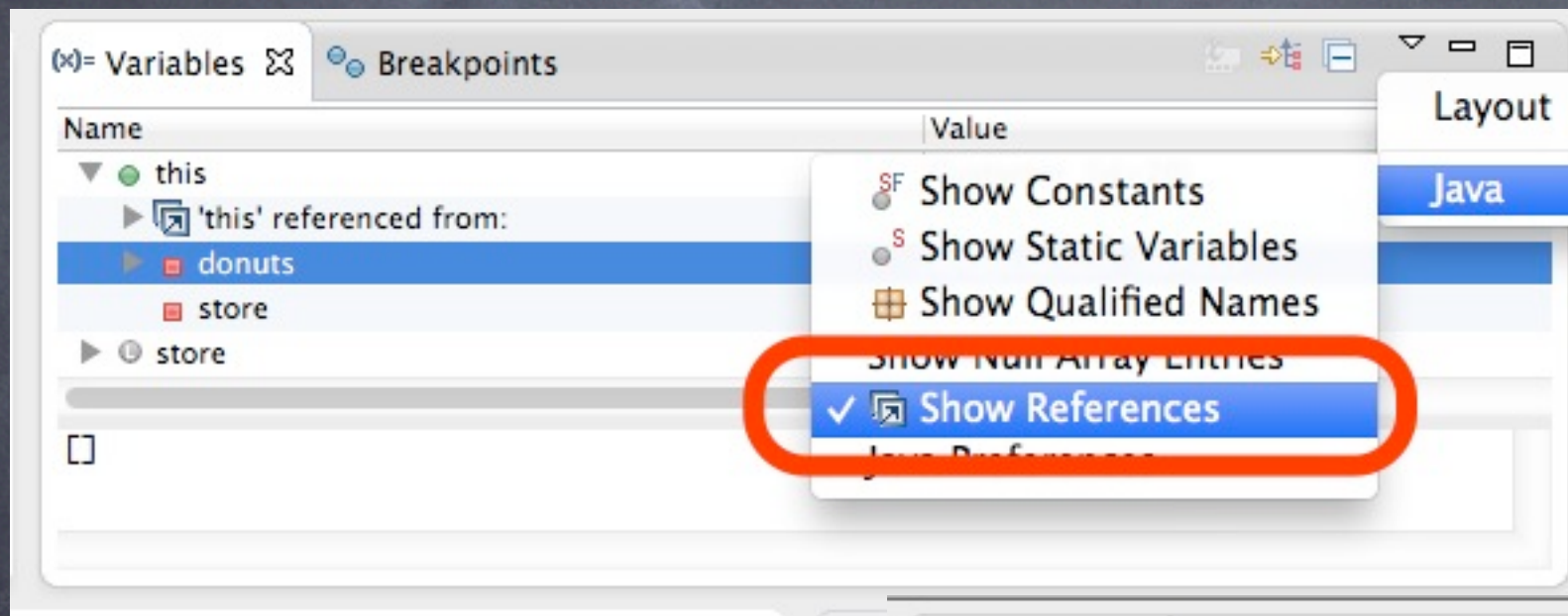
Breakpoint Properties...を選ぶ

Conditionalにチェック

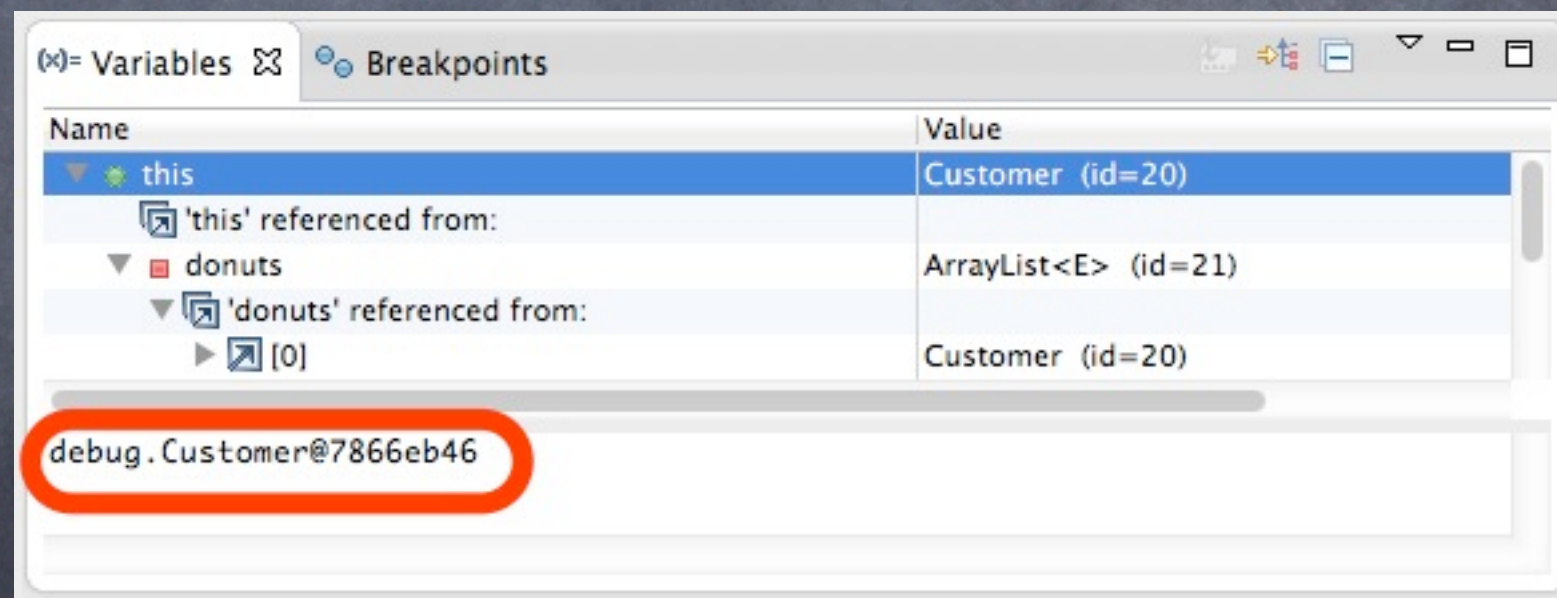
Suspend when value changesにチェックを入れ、変更を監視したい変数を入力しましょう。(nullからの変更も監視されます。)

Q24. 参照しているオブジェクトを見る方法はありませんか？

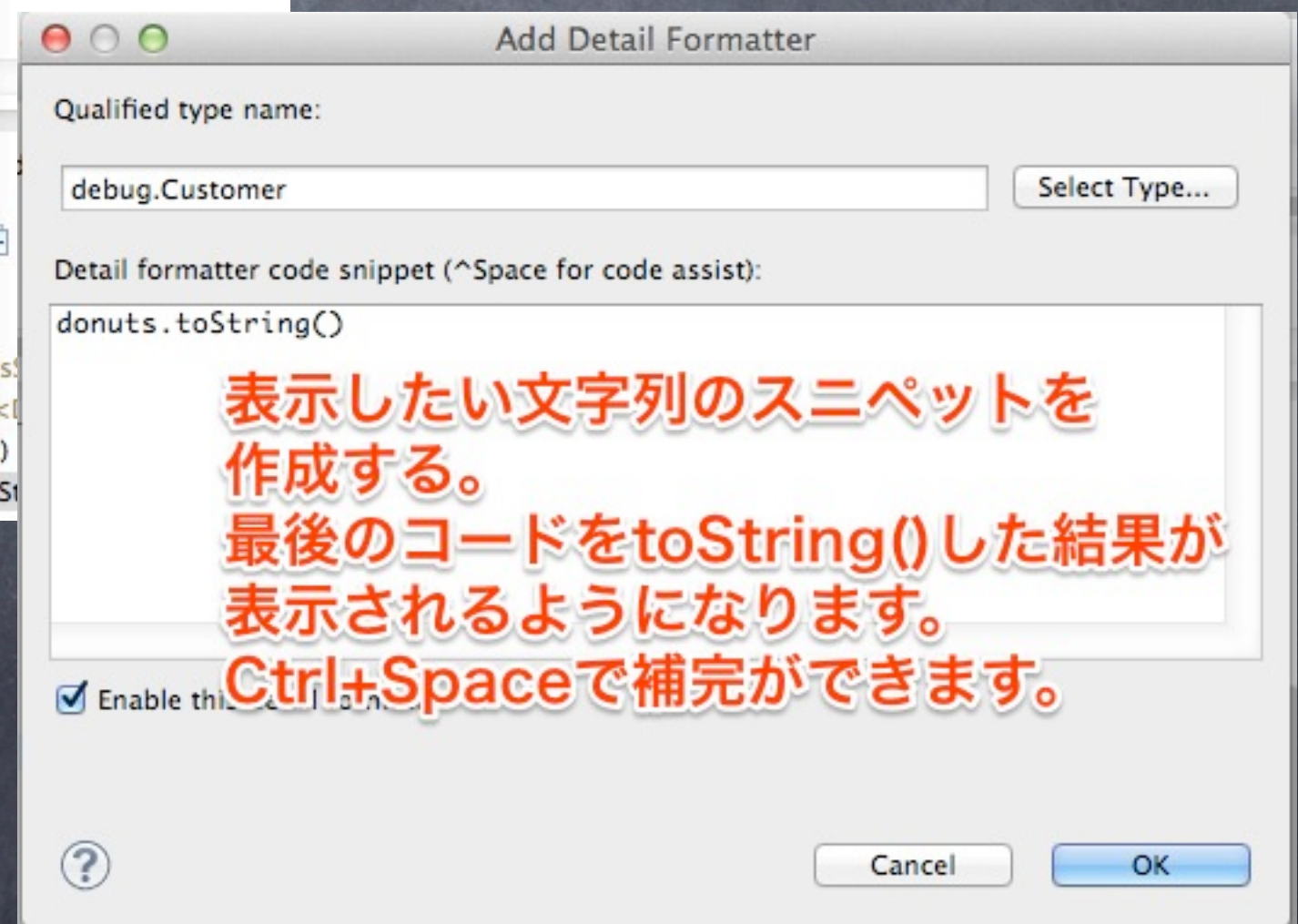
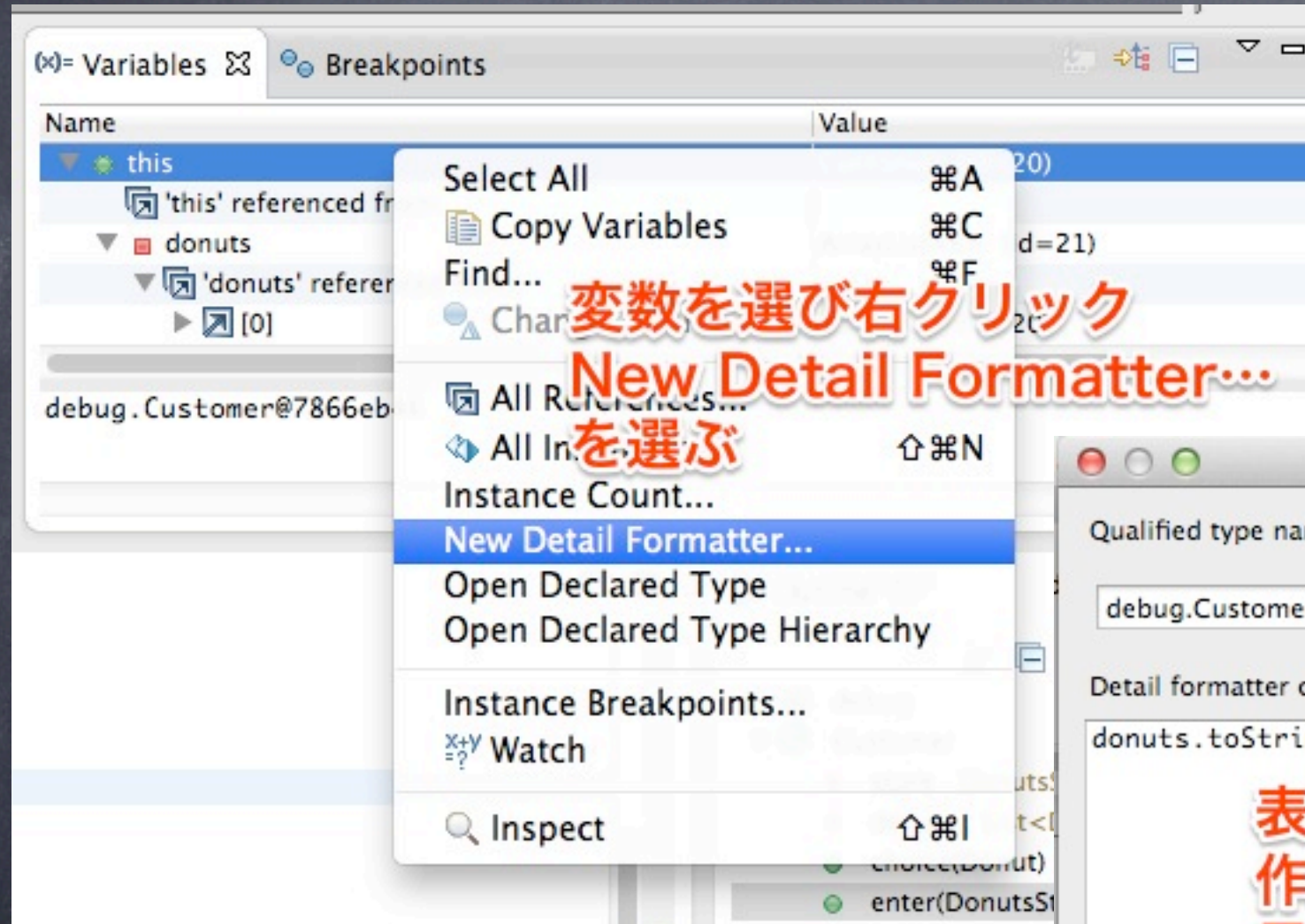
A. 変数ビューで「Show References」を選びましょう



Q25. 変数ビューの表示が
分かりやすく変更できません
か？



A. Detail Formatterを使いましょう。



Q26. sun.*やjava.*等の
パッケージのクラスには
Step Inしたくありません。
飛ばす方法はありません
か？

A. Step Filteringを使いま しょう。

デバッグビューで右クリック
Edit Step Filters... を選択

チェックを入れた
クラスや
パッケージには
StepInしなく
なります

チェック

Step Filtering

Step filters are applied when the 'Use Step Filters' toggle is is

☒ Use Step Filters

Defined step filters:

- ☐ com.ibm.*
- ☐ com.sun.*
- ☐ java.*
- ☐ javax.*
- ☐ jrockit.*
- ☐ org.omg.*
- ☐ sun.*
- ☐ sunw.*
- ☒ java.lang.ClassLoader

☐ Filter synthetic methods (requires VM support)

☐ Filter static initializers

☐ Filter constructors

☐ Filter simple getters

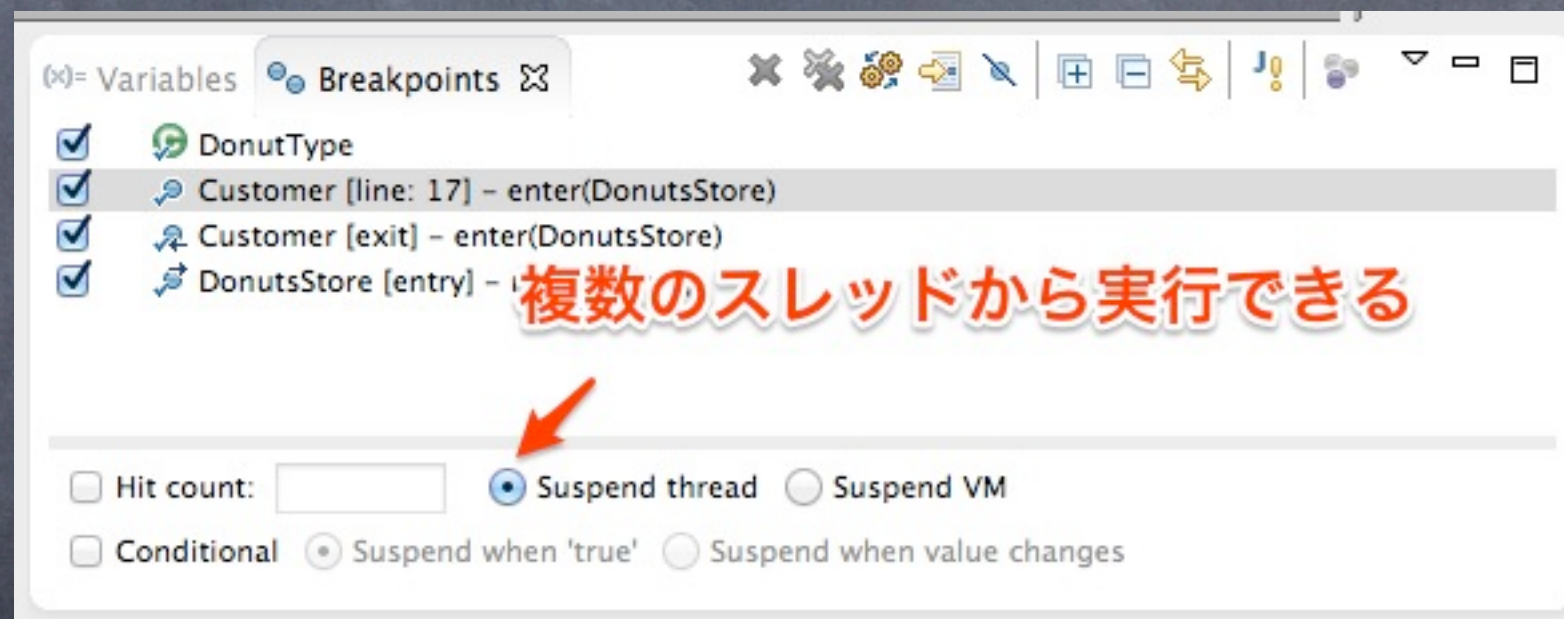
☐ Filter simple setters

☒ Step through filters

Restore Defaults

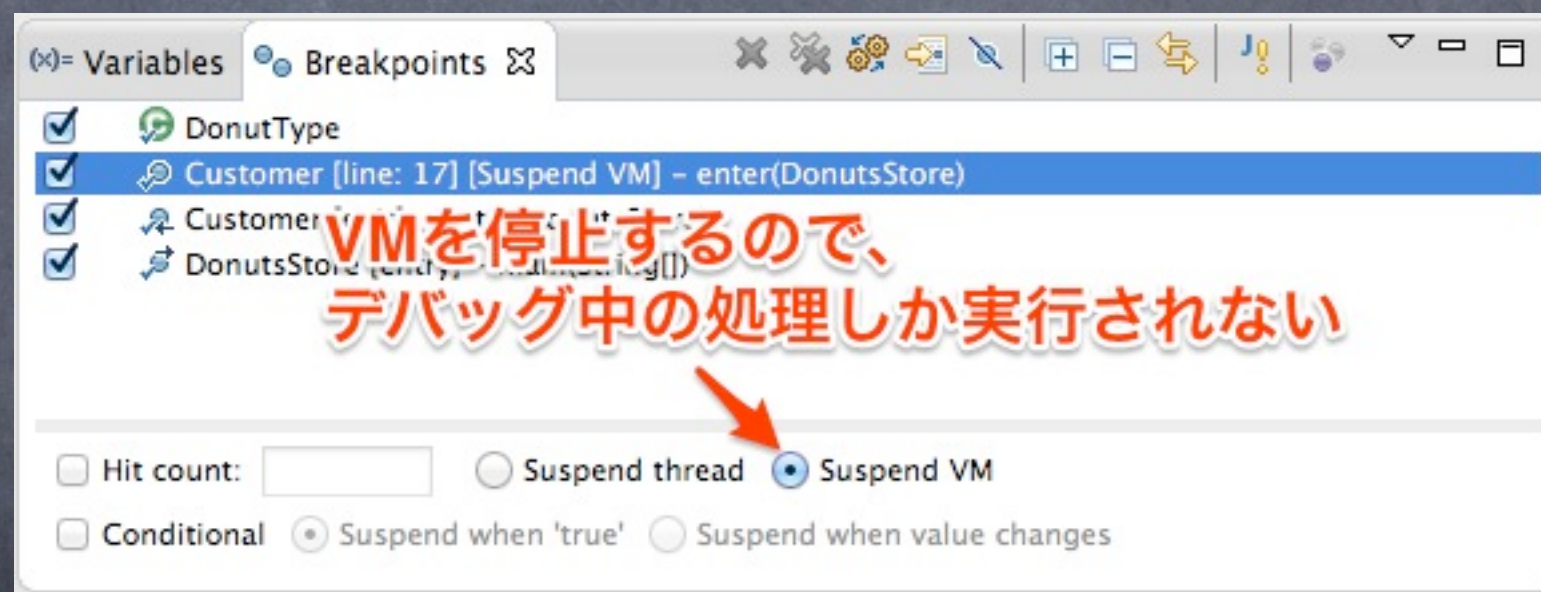
Q27. 複数のスレッドによる
競合のデバッグをするに
はどうすればよいです
か？

A. スレッド停止を選択し、複数の処理を同時にデバッグしましょう。



Q28. バックグラウンドで行われる処理がデバッグを邪魔します。どうすればデバッグしている処理だけに限定できますか??

A. VM停止を選択し、デバッグ中は他の処理を停止しましょう。



Q29. 起動引数を知る方法
はありますか？

A. デバッグビューから起動 引数を調べられます。

The screenshot shows the Eclipse IDE interface. In the top-left, the 'Debug' view is open, displaying a tree of debug elements. A red arrow points to the 'Thread [main] (Suspended (exit of method e...))' entry. Below this, the 'Properties' dialog is open, showing the 'Process properties' tab. The 'Command Line' field is highlighted with a red rectangle, showing the command used to start the application. The 'Environment' field is empty.

1.調べたいプロセスを選択
2.右クリック→Properties

```
DonutsStore.java Donut.java DonutStore.java  
this.donuts.add(donut);  
}  
public void enter(DonutsStore store)  
{  
    this.store = store;  
    if(store == null){  
        System.out.println("store is null");  
    }  
    return;  
} else {  
    store.welcome();  
}  
}  
public void buy() {
```

Process properties

Run-at time:
2013/09/28 19:05:33

Path:
/Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/Contents/Home/bin/java

Working Directory:
/Users/kompiro/codes/java/debug-on-eclipse/debug

Command Line:
/Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/Contents/Home/bin/java
-agentlib:jdwp=transport=dt_socket,suspend=y,address=localhost:49262
-Dfile.encoding=UTF-8
-classpath
/Users/kompiro/codes/java/debug-on-eclipse/debug/bin
debug.DonutsStore

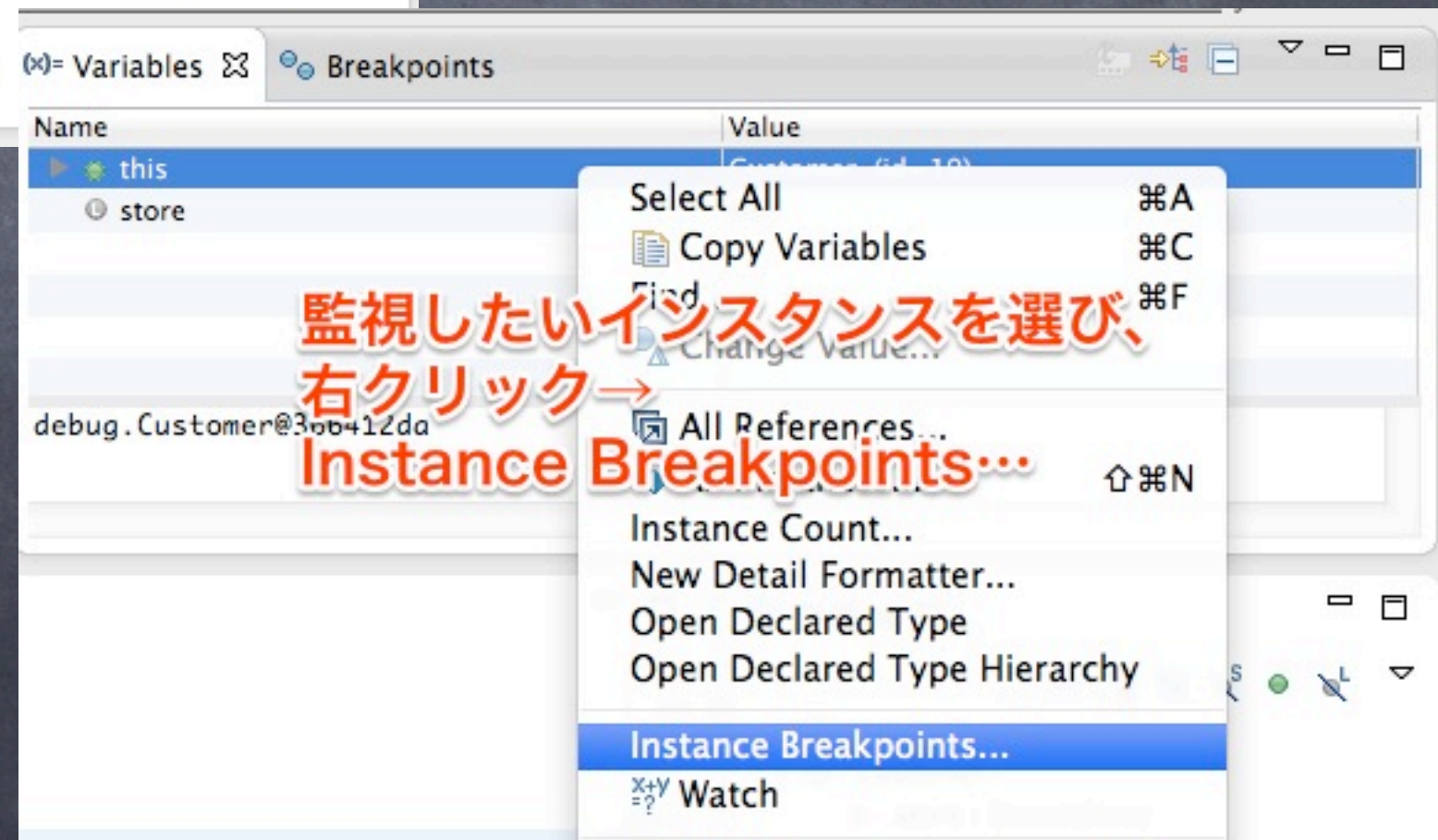
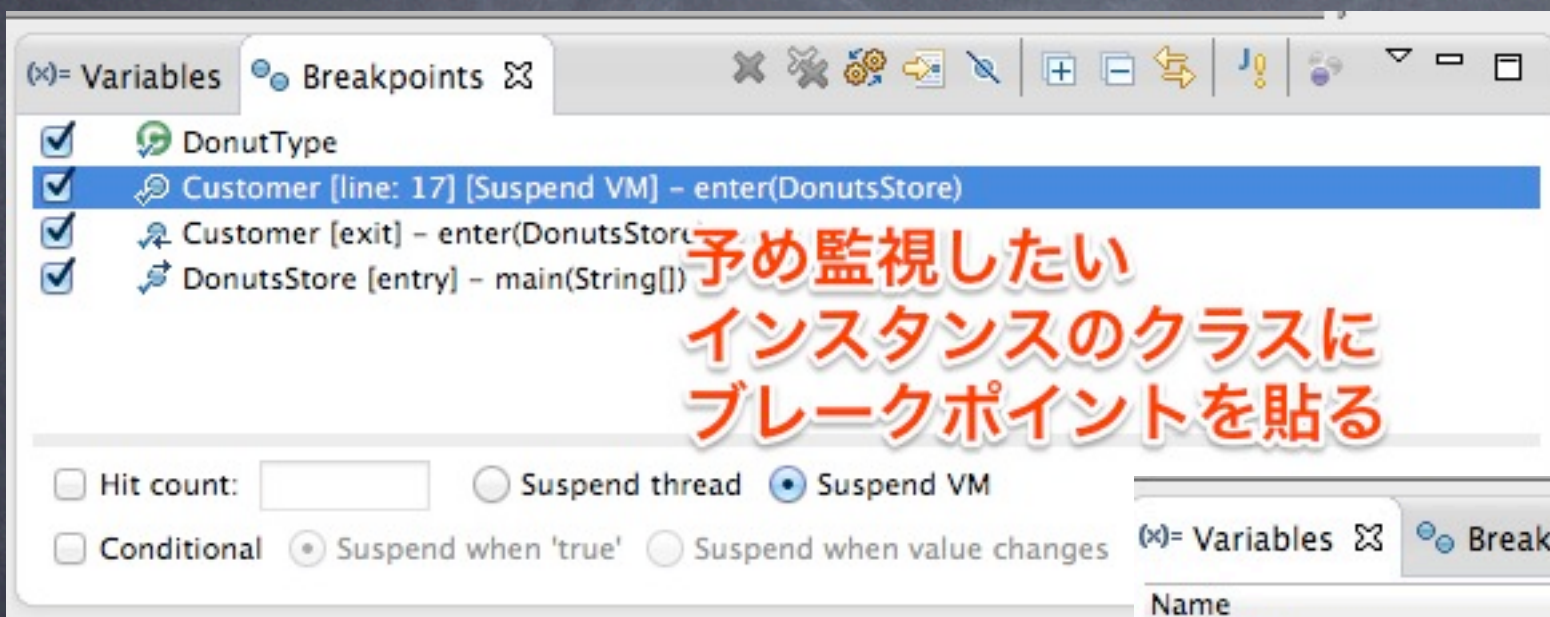
Environment:
No environment information provided

Cancel OK

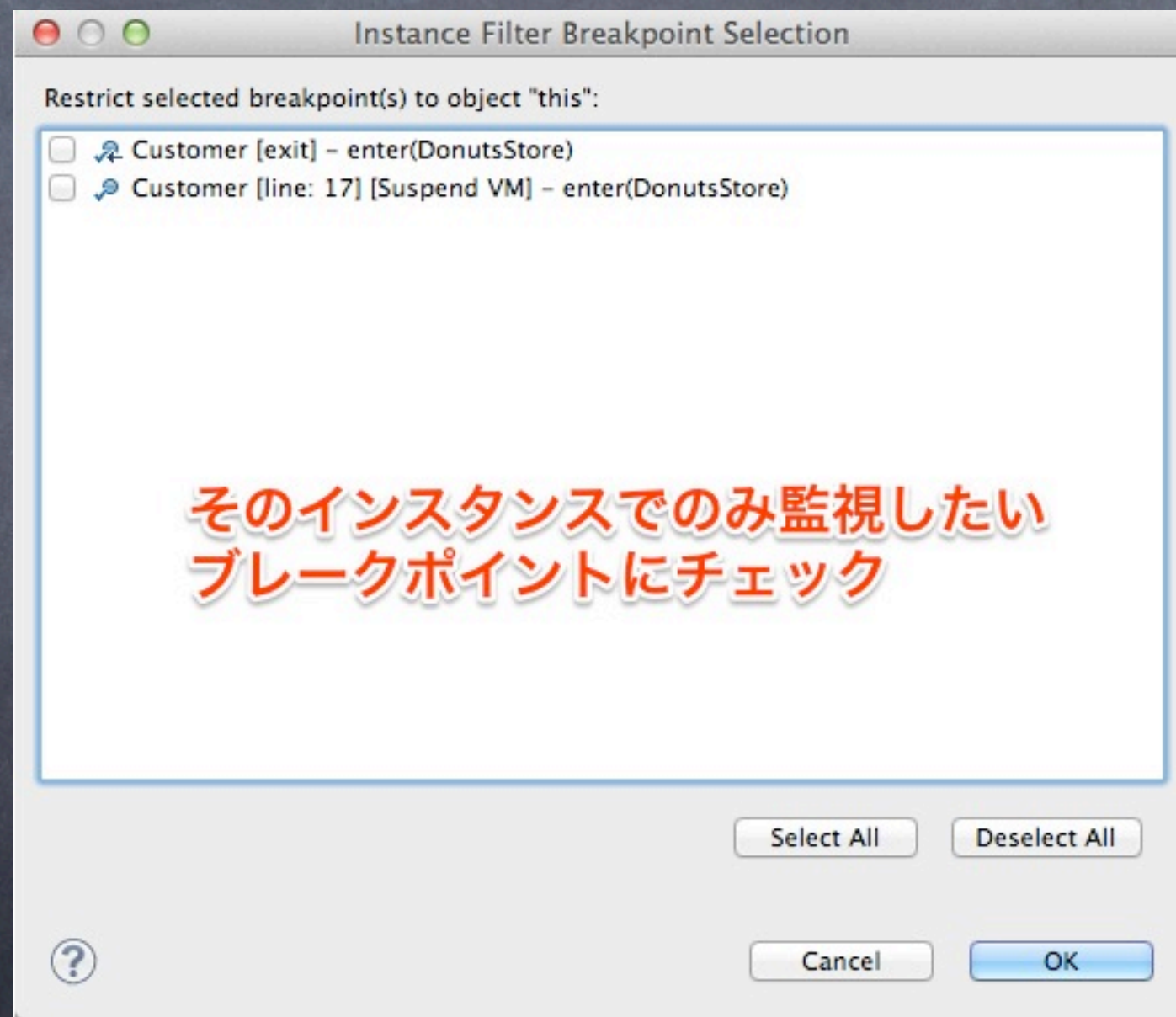
Q30. 特定のインスタンス
に注目してブレークポイント
を貼ることはできません
か？

Thanks @matobaa

A. 変数ビューのインスタンスからブレークポイントを指定できます。

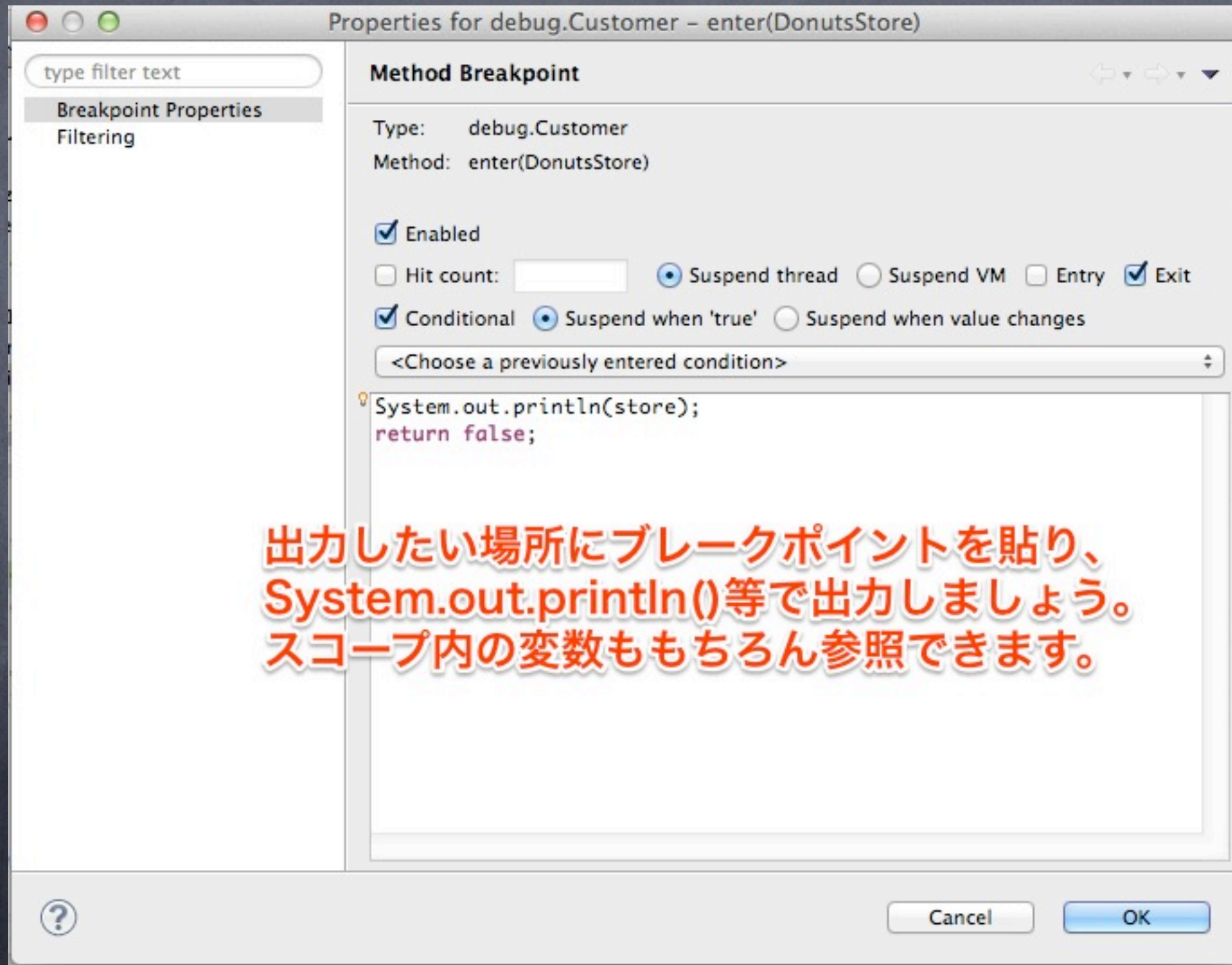


A. 変数ビューのインスタンスからブレークポイントを指定できます。



Q31. ソースコードを修正することなく、コンソールに値を出力することはできますか？

A. できます



おしまい

- 誤字脱字、もっとこうしたらいい等、フィードバックがあればkompiro@gmail.comにお寄せください。

謝辞

- フィードバックをお寄せくださった方(スライド順)
- @shuji_w6e, @yujiorama, @n3104, @matobaa