TRSpec Plugin User's Manual

March 2015

Table of Contents

1 Introduction

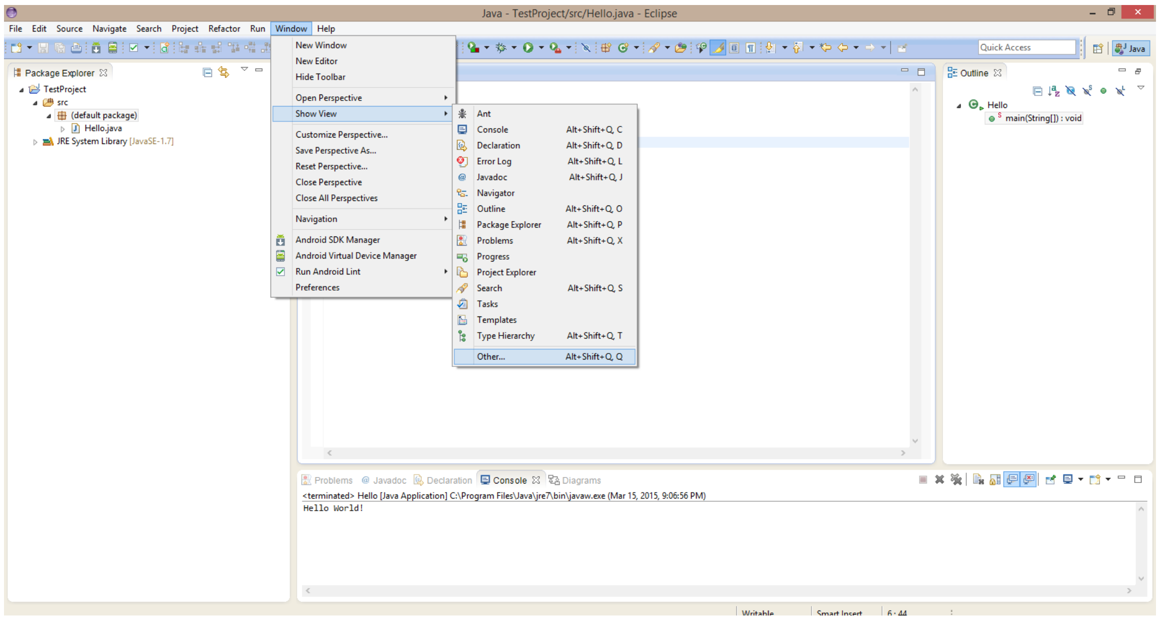
This document describes how to install and use the TRSpec plug-in for Eclipse.

2 Installing TRSpec

To install TRSpec place the TRSpec jar file in the *dropins* folder of the root directory of your Eclipse installation. Restart your Eclipse application and TRSpec will be available to use.

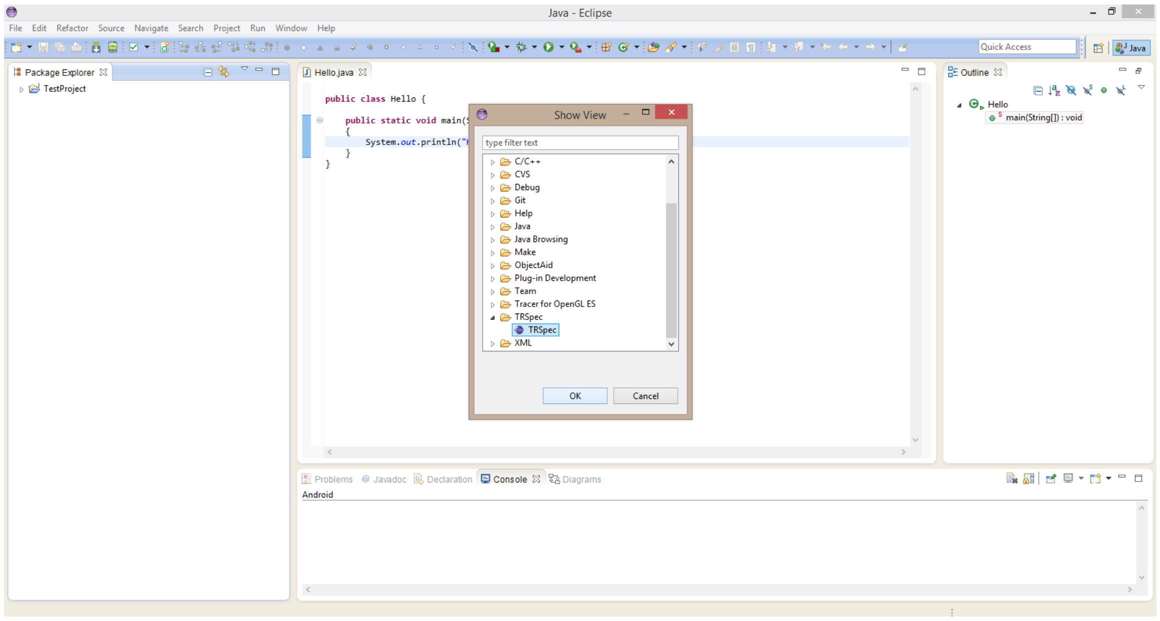
3 How to use

To open TRSpec, navigate to *Window→Show View→Other...*



```
<terminated> Hello [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 15, 2015, 9:06:56 PM)
Hello World!
```

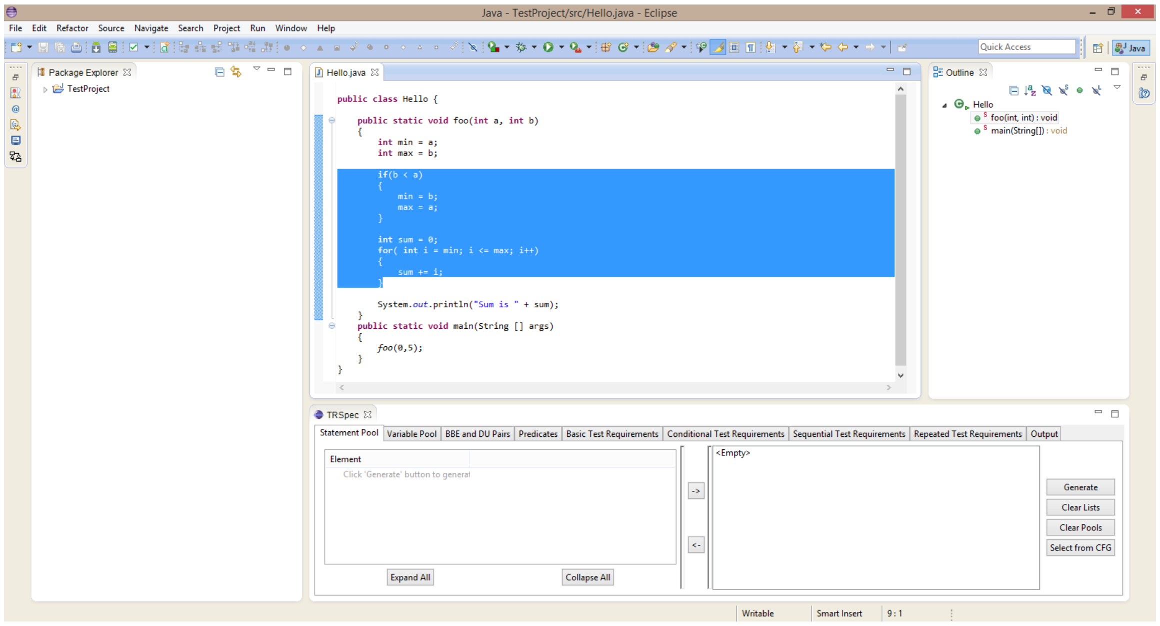
Then navigate to the *TRSpec*folder, highlight *TRSpec*, and press OK.



```
public class Hello {

    public static void main(S
    {
        System.out.println("H
    }
}
```

Selecting code

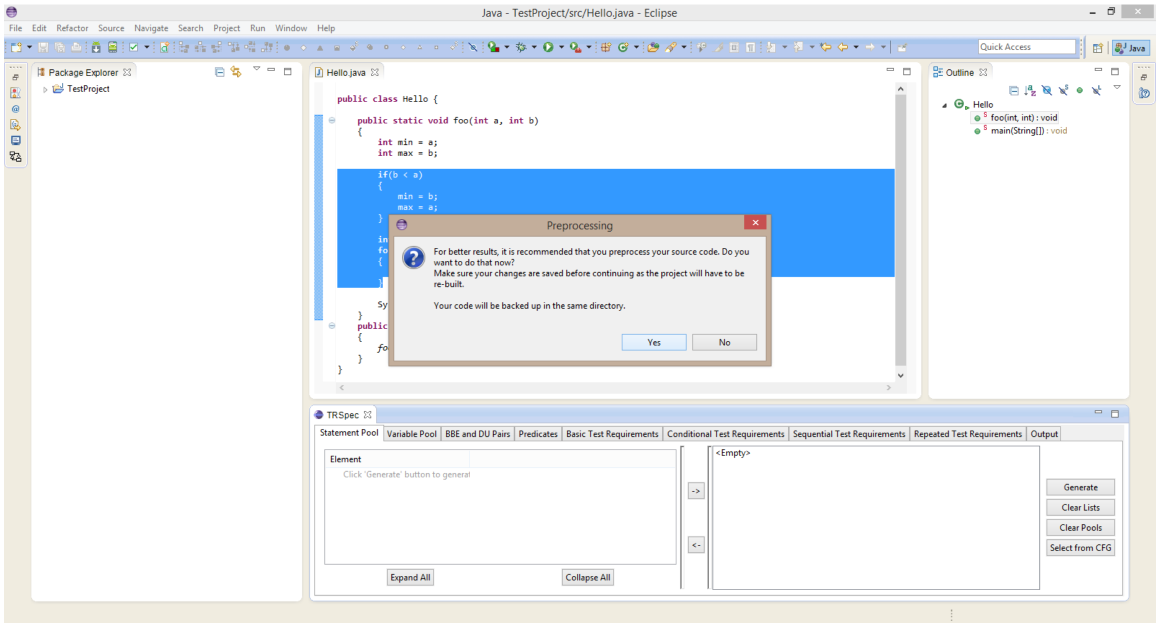
To begin using TRSpec, first highlight the Java code that you wish to derive bytecode statements and variables from.



```
public class Hello {

    public static void foo(int a, int b)
    {
        int min = a;
        int max = b;

        if(b < a)
        {
            min = b;
            max = a;
        }

        int sum = 0;
        for( int i = min; i <= max; i++)
        {
            sum += i;
        }

        System.out.println("Sum is " + sum);
    }
    public static void main(String [] args)
    {
        foo(0,5);
    }
}
```

Statement and Variable Pool

The process of specifying test requirements starts with the selection of the underlying basic constituents, namely bytecode statements and variables. Such elementary step is carried out in the *Statement Pool* and *Variable Pool* tabs. Each of these tabs is divided into two composites, a *list* composite (left section) and a *pool* composite (right section). After highlighting the Java code, click on the *Generate* button.

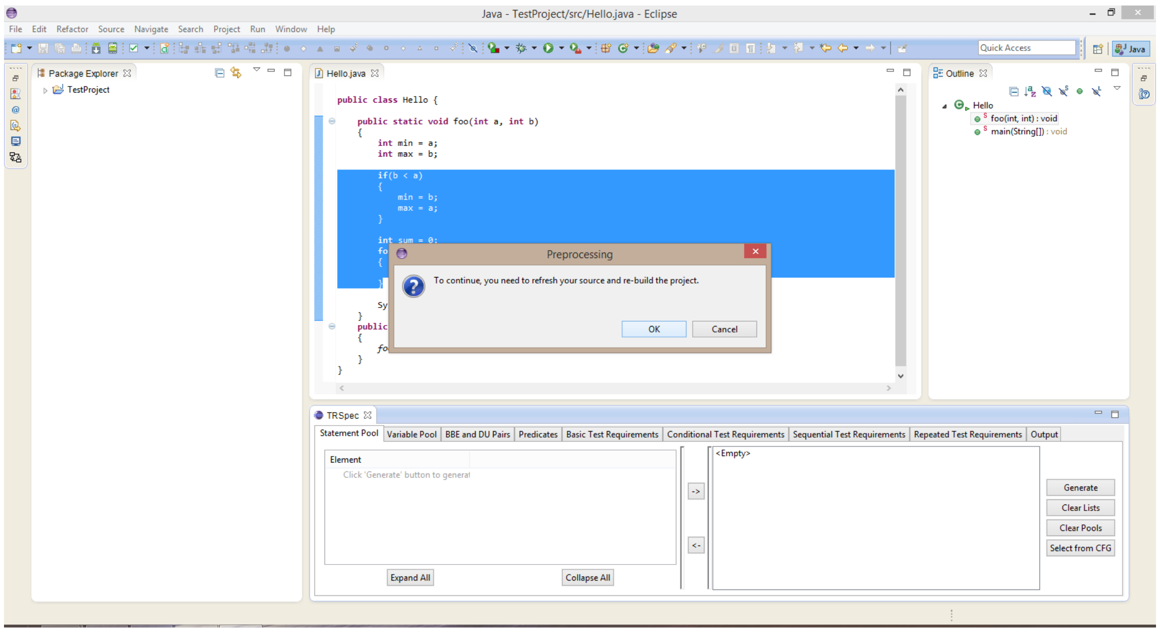


```
public class Hello {

    public static void foo(int a, int b)
    {
        int min = a;
        int max = b;

        if(b < a)
        {
            min = b;
            max = a;
        }

        in
        fo
        {

        }

        Sy
    }
    public
    {
        fo
    }
}
```

For better results, it is recommended that you preprocess your source code. Do you want to do that now?
Make sure your changes are saved before continuing as the project will have to be re-built.

Your code will be backed up in the same directory.

A pop-out will show asking you to pre-process your source code. Click Yes after making sure your code was saved.


Java - TestProject/src/Hello.java - Eclipse

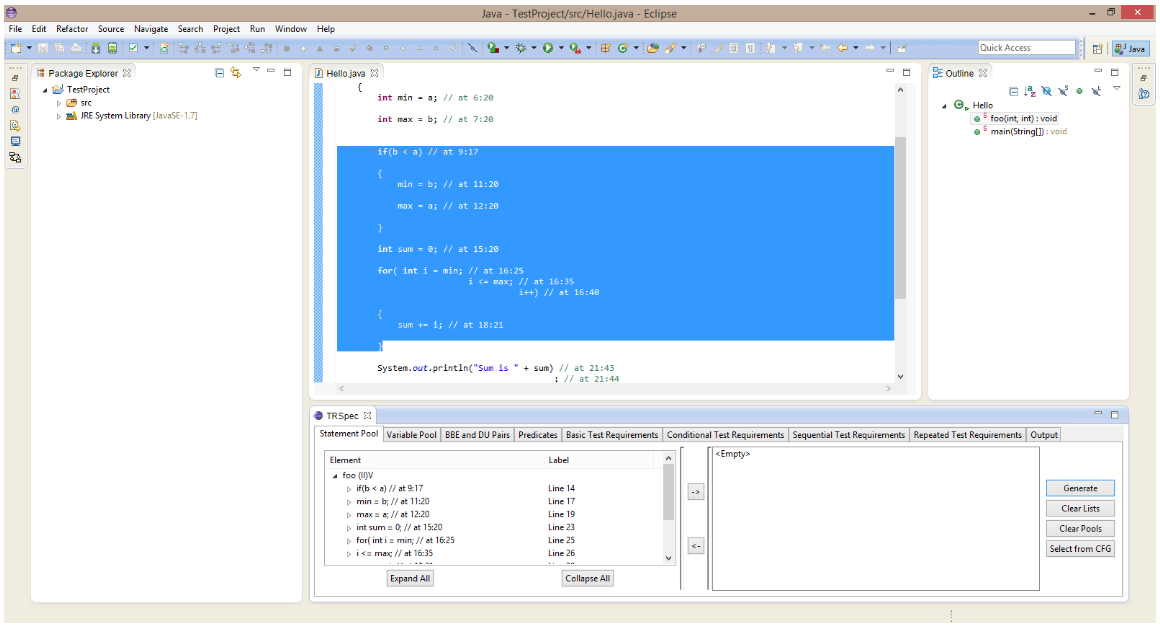
```
public class Hello {

    public static void foo(int a, int b)
    {
        int min = a;
        int max = b;

        if(b < a)
        {
            min = b;
            max = a;
        }

        int sum = 0;
        fo
        {

        }

        Sy
    }
    public
    {
        fo
    }
}
```

Preprocessing

To continue, you need to refresh your source and re-build the project.

OK Cancel

Another pop-out will show asking you to refresh and re-build your project. Click on Ok.

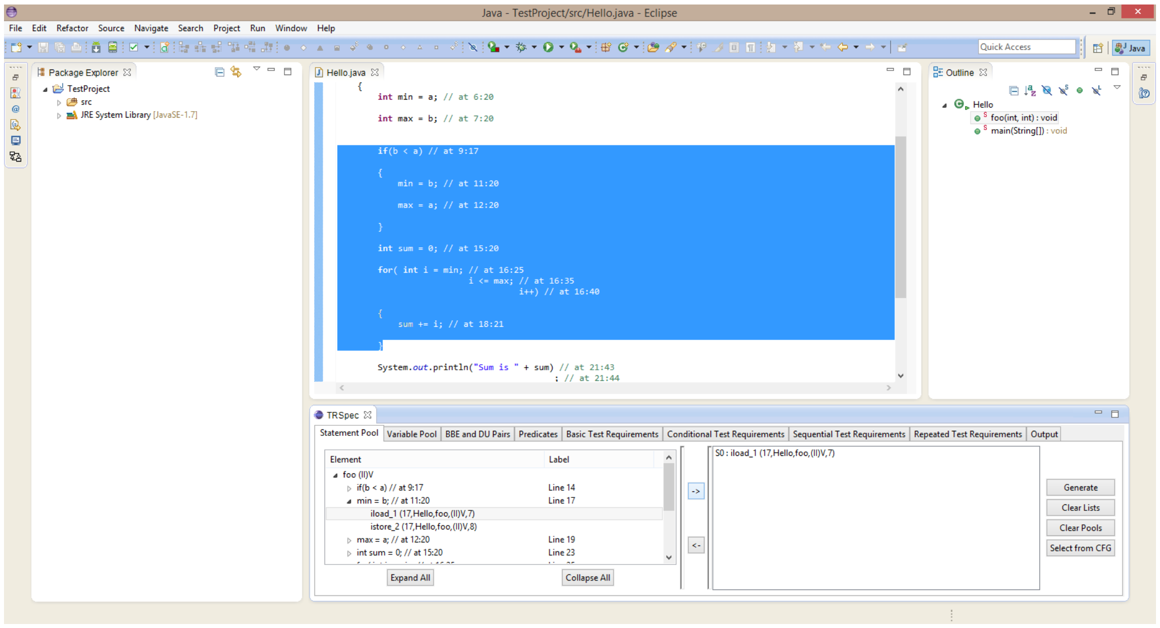

Java - TestProject/src/Hello.java - Eclipse

```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                     i <= max; // at 16:35
                           i++) // at 16:40

        {
            sum += i; // at 18:21

        }

        System.out.println("Sum is " + sum) // at 21:43
                                      ; // at 21:44
```

Element	Label
foo (II)V	
if(b < a) // at 9:17	Line 14
min = b; // at 11:20	Line 17
max = a; // at 12:20	Line 19
int sum = 0; // at 15:20	Line 23
for(int i = min; // at 16:25	Line 25
i <= max; // at 16:35	Line 26

<Empty>

This will generate all the bytecode statements and variables associated with the code you highlighted; these will be inserted in the corresponding lists. To insert a statement into the *statements pool*, navigate to the *statements list*, highlight the desired statement and click on the→ button adjacent to it.


Java - TestProject/src/Hello.java - Eclipse

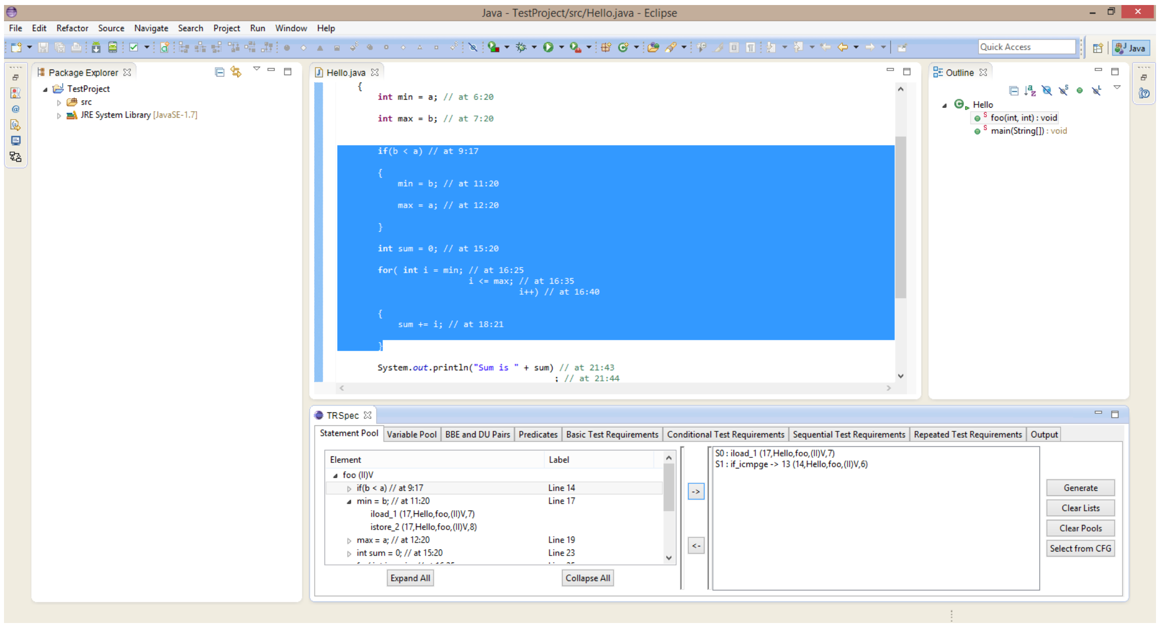
```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                      i <= max; // at 16:35
                           i++) // at 16:40

        {
            sum += i; // at 18:21

        }

        System.out.println("Sum is " + sum) // at 21:43
                                  ; // at 21:44
```

TRSpec — Statement Pool

Element	Label
foo (II)V	
if(b < a) // at 9:17	Line 14
min = b; // at 11:20	Line 17
iload_1 (17,Hello,foo,(II)V,7)	
istore_2 (17,Hello,foo,(II)V,8)	
max = a; // at 12:20	Line 19
int sum = 0; // at 15:20	Line 23

```
S0 : iload_1 (17,Hello,foo,(II)V,7)
```

Or you can just click on your java line from the listand click on the→ button and the last byte code statement of that line will be inserted to the *statements pool*.


Java - TestProject/src/Hello.java - Eclipse

```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                      i <= max; // at 16:35
                           i++) // at 16:40

        {
            sum += i; // at 18:21

        }

        System.out.println("Sum is " + sum) // at 21:43
                                  ; // at 21:44
```

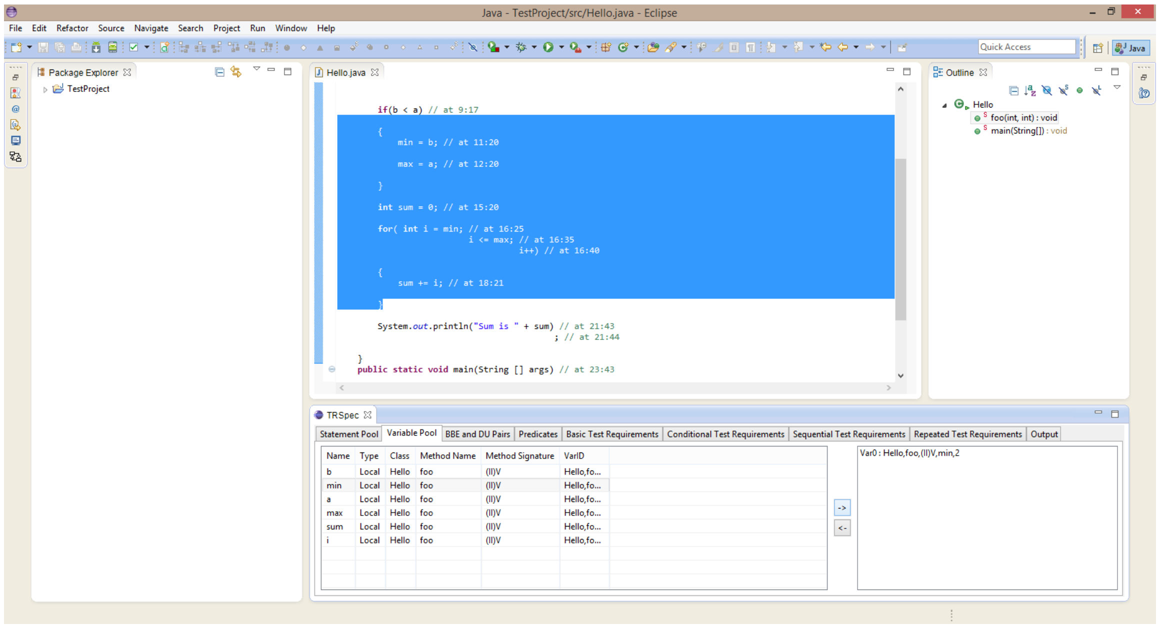
TRSpec — Statement Pool

Element	Label
foo (II)V	
if(b < a) // at 9:17	Line 14
min = b; // at 11:20	Line 17
iload_1 (17,Hello,foo,(II)V,7)	
istore_2 (17,Hello,foo,(II)V,8)	
max = a; // at 12:20	Line 19
int sum = 0; // at 15:20	Line 23

```
S0 : iload_1 (17,Hello,foo,(II)V,7)
S1 : if_icmpge -> 13 (14,Hello,foo,(II)V,6)
```

To remove a statement from the pool, simply highlight it and click on the ←button.

To insert avariable into the *variables pool*, click on *VariablePool*tab and navigate to the *variables list*, highlight the desired variable and click on the→ button adjacent to it.



```
        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                      i <= max; // at 16:35
                            i++) // at 16:40

        {
            sum += i; // at 18:21

        }

        System.out.println("Sum is " + sum) // at 21:43
                                           ; // at 21:44

    }
    public static void main(String [] args) // at 23:43
```

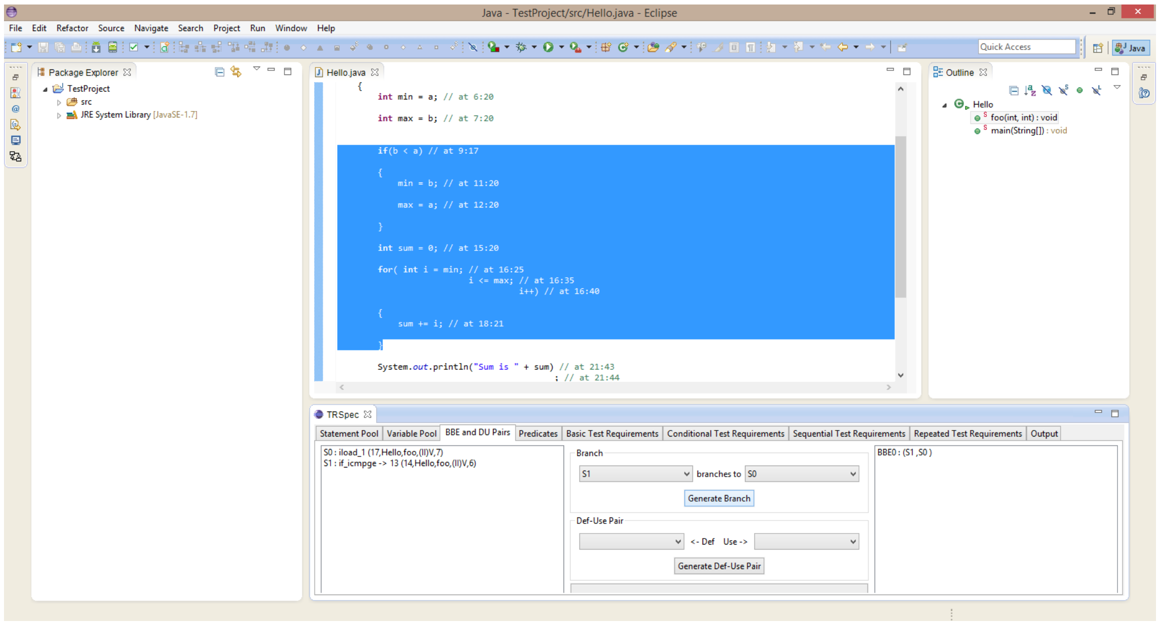
Name	Type	Class	Method Name	Method Signature	VarID
b	Local	Hello	foo	(II)V	Hello,fo...
min	Local	Hello	foo	(II)V	Hello,fo...
a	Local	Hello	foo	(II)V	Hello,fo...
max	Local	Hello	foo	(II)V	Hello,fo...
sum	Local	Hello	foo	(II)V	Hello,fo...
i	Local	Hello	foo	(II)V	Hello,fo...

Var0 : Hello,foo,(II)V,min,2

To remove a variable from the pool, simply highlight it and click on the ←button.

BBE and DU Pairs

To specify a branch (basic block edge or *BBE*), navigate to the branch composite, select the statement to branch from inthe combo on the left, select the statement to branch to in the combo on the right, and then click onthe *Generate Branch* button. Note that the options listed are managed in a way to make sure that the source and the target form a valid branch.



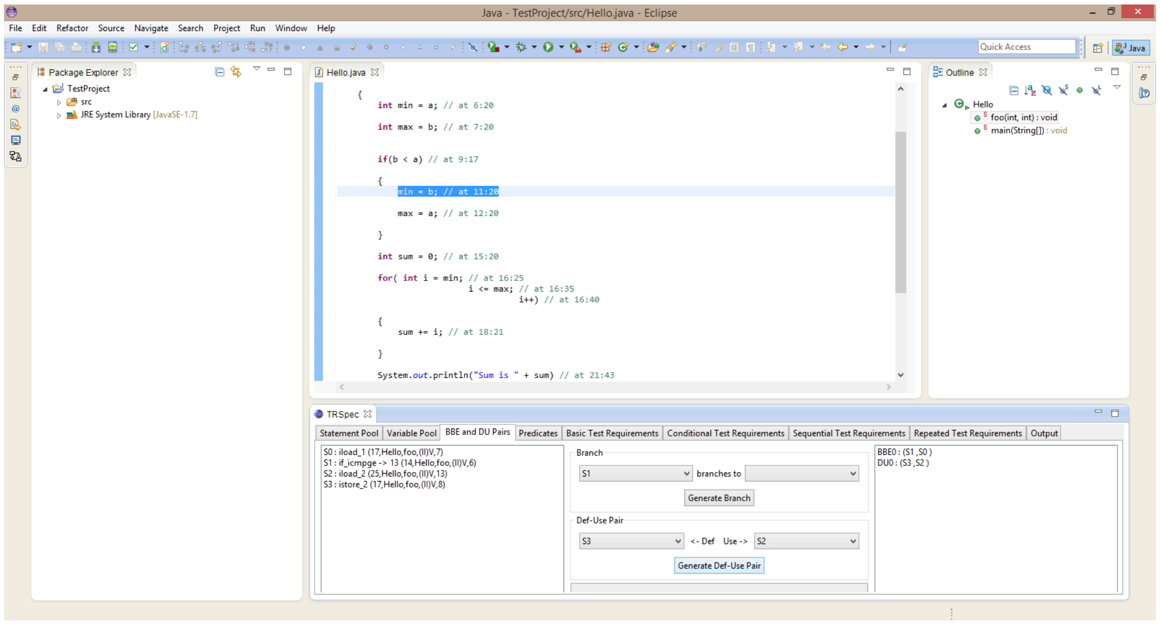
```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                     i <= max; // at 16:35
                           i++) // at 16:40

        {
            sum += i; // at 18:21

        }

        System.out.println("Sum is " + sum) // at 21:43
                                   ; // at 21:44
```

```
S0 : iload_1 (17,Hello,foo,(II)V,7)
S1 : if_icmpge -> 13 (14,Hello,foo,(II)V,6)
```

Branch: S1 branches to S0

```
BBE0 : (S1 ,S0 )
```

Specifying def-use pairs can be done similarly.


Java - TestProject/src/Hello.java - Eclipse
File Edit Refactor Source Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
TestProject
src
JRE System Library [JavaSE-1.7]
Hello.java
Outline
Hello
foo(int, int) : void
main(String[]) : void
TRSpec
Statement Pool | Variable Pool | BBE and DU Pairs | Predicates | Basic Test Requirements | Conditional Test Requirements | Sequential Test Requirements | Repeated Test Requirements | Output
Branch
branches to
Generate Branch
Def-Use Pair
<- Def Use ->
Generate Def-Use Pair

```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                      i <= max; // at 16:35
                             i++) // at 16:40

        {
            sum += i; // at 18:21

        }

        System.out.println("Sum is " + sum) // at 21:43
```

```
S0 : iload_1 (17,Hello,foo,(II)V,7)
S1 : if_icmpge -> 13 (14,Hello,foo,(II)V,6)
S2 : iload_2 (25,Hello,foo,(II)V,13)
S3 : istore_2 (17,Hello,foo,(II)V,8)
```

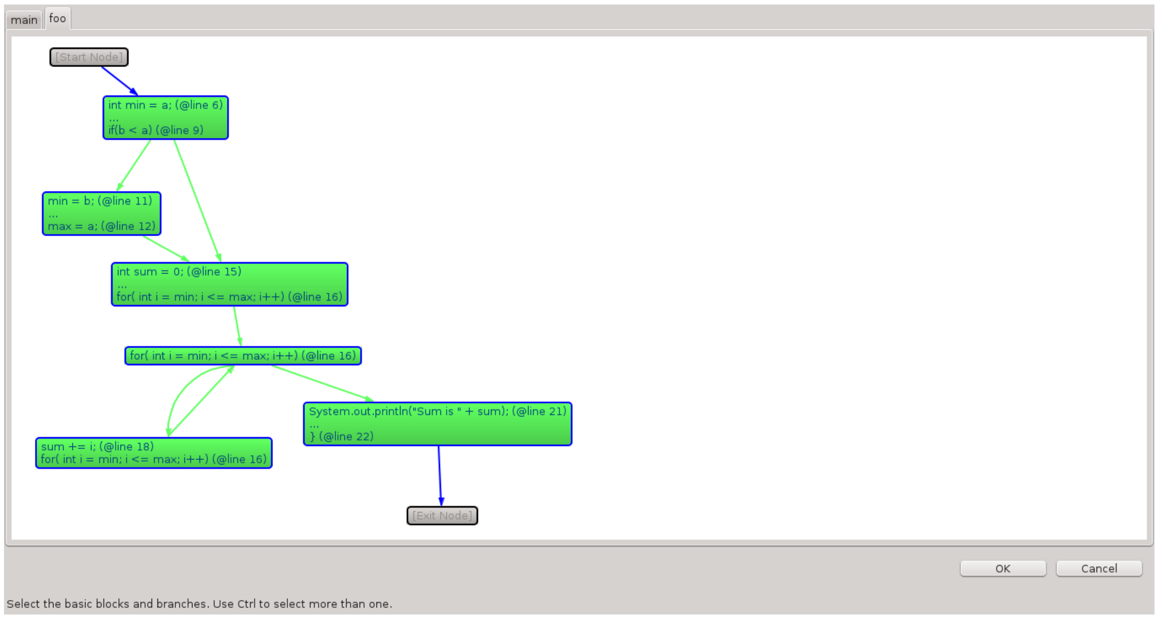
Branch: S1 branches to

Def-Use Pair: S3 <- Def Use -> S2

```
BBE0 : (S1 ,S0 )
DU0 : (S3 ,S2 )
```

Using the Control Flow Graph

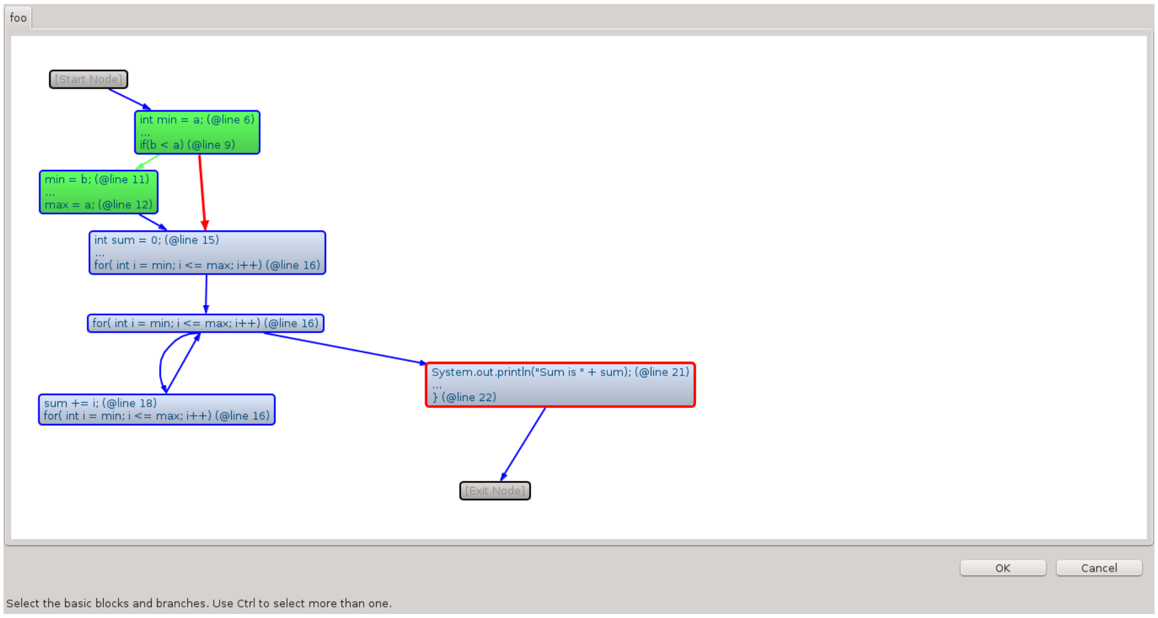
To simplify this process, it is also possible to select the needed elements from the Control Flow Graph (CFG) of your program. To do that, you can click on the “Select from CFG” button after selecting the relevant part of your code in the editor. You might be asked to generate the model for your code if you had not done that previously; if that is the case, press “Ok”. You will see a pop-up dialog that looks similar to this one:



Select the basic blocks and branches. Use Ctrl to select more than one.

Each of the Java methods you had selected will have its own tab in the dialog (in the above case “main” and “foo”), with separate independent CFGs for each method. The CFG has an entry and exit node (in gray), basic blocks (all other nodes), and branches (the connections of the graph). The basic blocks and branches are color coded. The meaning of each color is as follows:

- Basic blocks with a *green background* represent the code which was selected in the editor
- Branches colored in *green* represent the branches which are selected in the editor, i.e. the source and target basic blocks were selected in the editor
- Basic blocks and branches with a *blue border* are those which have not been added to their respective pools (in order to be used when defining the test requirements)
- Basic blocks and branches with a *red border* are those which have been added to their respective pools
- Selected items will be highlighted in *orange* and *yellow* to allow for easier visualization of the control flow



```
int min = a; (@line 6)
...
if(b < a) (@line 9)

min = b; (@line 11)
...
max = a; (@line 12)

int sum = 0; (@line 15)
...
for( int i = min; i <= max; i++) (@line 16)

for( int i = min; i <= max; i++) (@line 16)

sum += i; (@line 18)
for( int i = min; i <= max; i++) (@line 16)

System.out.println("Sum is " + sum); (@line 21)
...
} (@line 22)
```

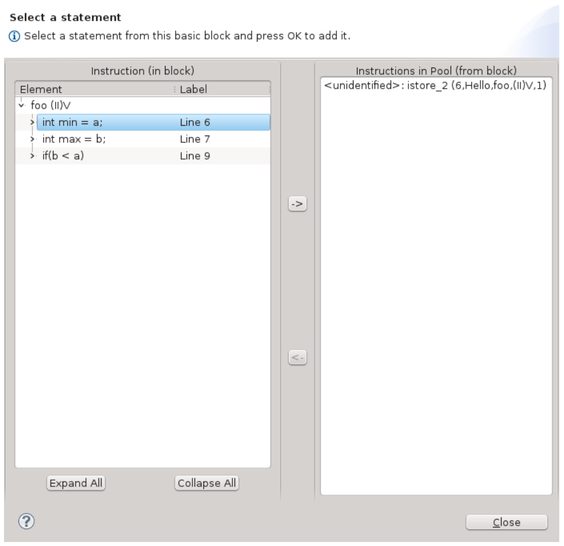
Select the basic blocks and branches. Use Ctrl to select more than one.

Hint: Each node shows the starting and ending lines of the Java code that corresponds to the basic block. Hovering the mouse over the block will show the full code for reference.

To use a basic block in a test requirement, you can double click on its node. This will add the last statement in this basic block to the *Statement pool*. By definition, if any statement in the basic block is executed, all the statements are supposed to execute (unless exceptions occur – more about this later). Hence, if any statement is selected within the block it will be marked with a red border. If you wish to select a specific statement within that basic block, you can always right click on it and select “Select individual statement”. A pop-up will show; it allows you to add and remove statements from the *Statement pool*. For details about its usage, refer to the “Statement and Variable Pool” section.

To use a branch in a test requirement, it can be added by double clicking a connection. Each connection represents a branch from the end of one basic block to the start of the next block. Similarly, to de-select a branch, you can double click a connection marked in red.

To use a def-use in a test requirement, it can also be added from the CFG. To do that, you need to identify in which basic block the def is found. Then, you can right click on that basic block and select "Select DU-Pair". A pop-up will show with the list of *definitions* in the top left corner and a list of its *corresponding uses* in the whole method. To include one in your test requirement, highlight the relevant def and use Java statements (or bytecode statements if more precision is needed), then press the adjacent → button. If it is no longer needed, you can remove it by pressing the arrow pointing to the other direction.



Select a statement

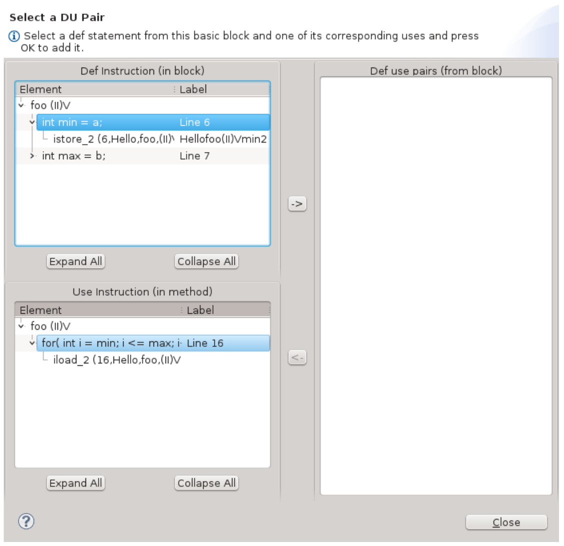
Select a statement from this basic block and press OK to add it.

Instruction (in block)

Element	Label
foo (II)V	
int min = a;	Line 6
int max = b;	Line 7
if(b < a)	Line 9

Instructions in Pool (from block)

```
<unidentified>: istore_2 (6,Hello,foo,(II)V,1)
```



Select a DU Pair

Select a def statement from this basic block and one of its corresponding uses and press OK to add it.

Def Instruction (in block)

Element	Label
foo (II)V	
int min = a;	Line 6
istore_2 (6,Hello,foo,(II)\	Hellofoo(II)Vmin2
int max = b;	Line 7

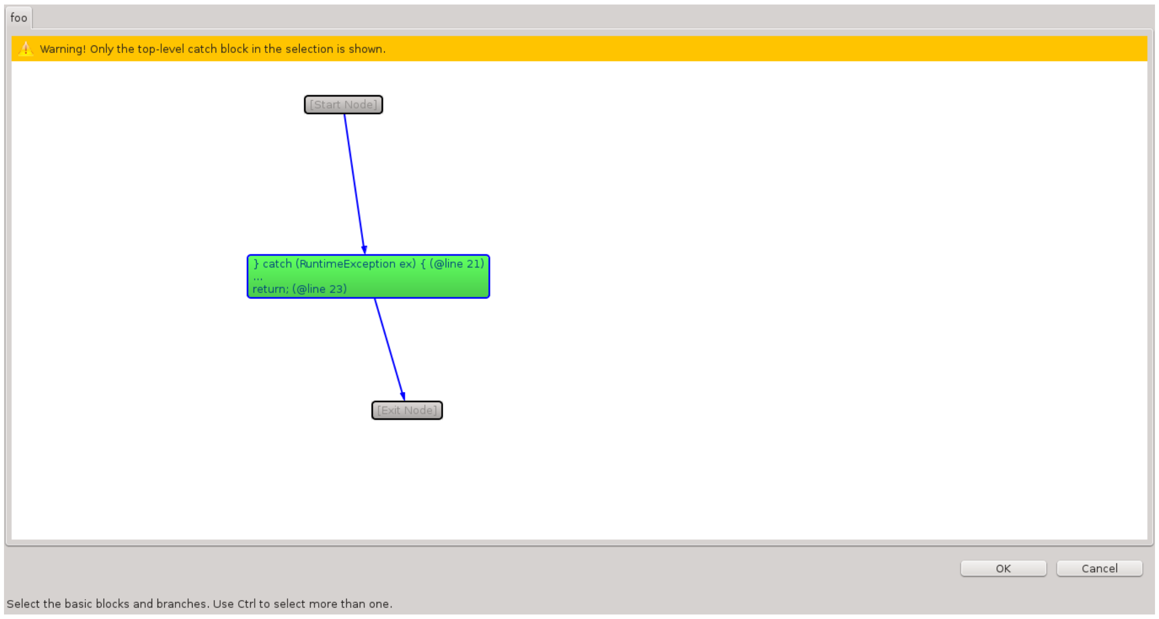
Use Instruction (in method)

Element	Label
foo (II)V	
for(int i = min; i <= max; i·	Line 16
iload_2 (16,Hello,foo,(II)V	

Def use pairs (from block)

Once all the necessary selections are done, press Ok to confirm your changes. You can also press Cancel if you want to discard your changes. You can now continue with the next steps.

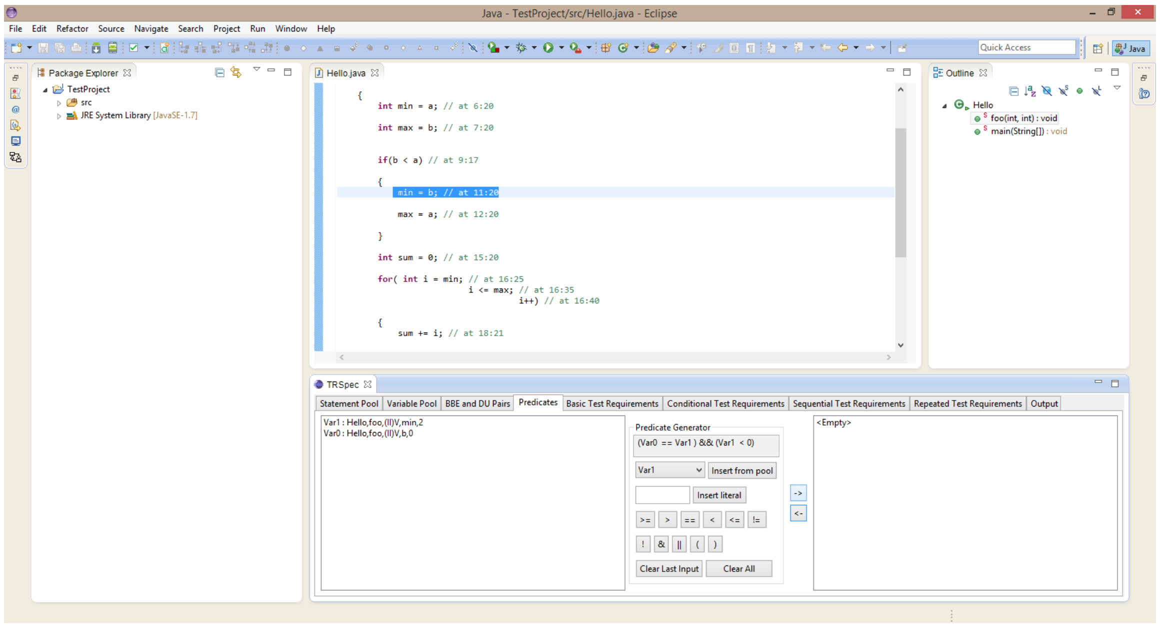
A note on exceptions and try-catch blocks: If the selected code in the editor contains a try-catch block, it will not be displayed because it usually does not belong to the desired control flow and you will be warned about it. However, in some instances it can be important to cover an important element in a catch block (or a finally block). To do that, you will have to explicitly select only the relevant catch block in question, press "Generate" then "Select from CFG", and its CFG will be displayed alone, with its entry and exit nodes being the start of the catch or finally block and the exit node of the method respectively.


foo
Warning! Only the top-level catch block in the selection is shown.
[Start Node]
} catch (RuntimeException ex) { (@line 21)
...
return; (@line 23)
[Exit Node]
OK
Cancel

Select the basic blocks and branches. Use Ctrl to select more than one.

Predicates

1. Predicates are constructed using the variables selected in the *Pools* tab. They can either be atomic themselves or composed by joining other atomic predicates usingboolean operators (&&, ||, and !). Atomic predicates are specified by selecting two variables (or a variable and a literal) and a corresponding relational operator (>=, >, ==, <, <=, and !=).
    a. You may insert variables from the variable pool by selecting them in the combo and clicking on the *Insert from pool* button.
    b. You may insert literals by typing them into the corresponding textbox and clicking on the *Insert literal* button.
    c. You may insert boolean/relational operators by clicking on the respective buttons.
2. To clear the last input to the predicate, click on the *Clear Last Input* button. To clear the whole predicate, click on the *Clear All* button.
3. To insert the finished predicate into the pool, click on the → button. To remove a predicate from the pool, highlight it in the list on the right and click on the ← button.


Java - TestProject/src/Hello.java - Eclipse
File Edit Refactor Source Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
TestProject
src
JRE System Library [JavaSE-1.7]
Hello.java
Outline
Hello
foo(int, int) : void
main(String[]) : void
TRSpec
Statement Pool | Variable Pool | BBE and DU Pairs | Predicates | Basic Test Requirements | Conditional Test Requirements | Sequential Test Requirements | Repeated Test Requirements | Output
Predicate Generator
Insert from pool
Insert literal
->
<-
>= > == < <= !=
! & || ( )
Clear Last Input
Clear All

```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                      i <= max; // at 16:35
                            i++) // at 16:40

        {
            sum += i; // at 18:21
```

Var1 : Hello,foo,(II)V,min,2
Var0 : Hello,foo,(II)V,b,0

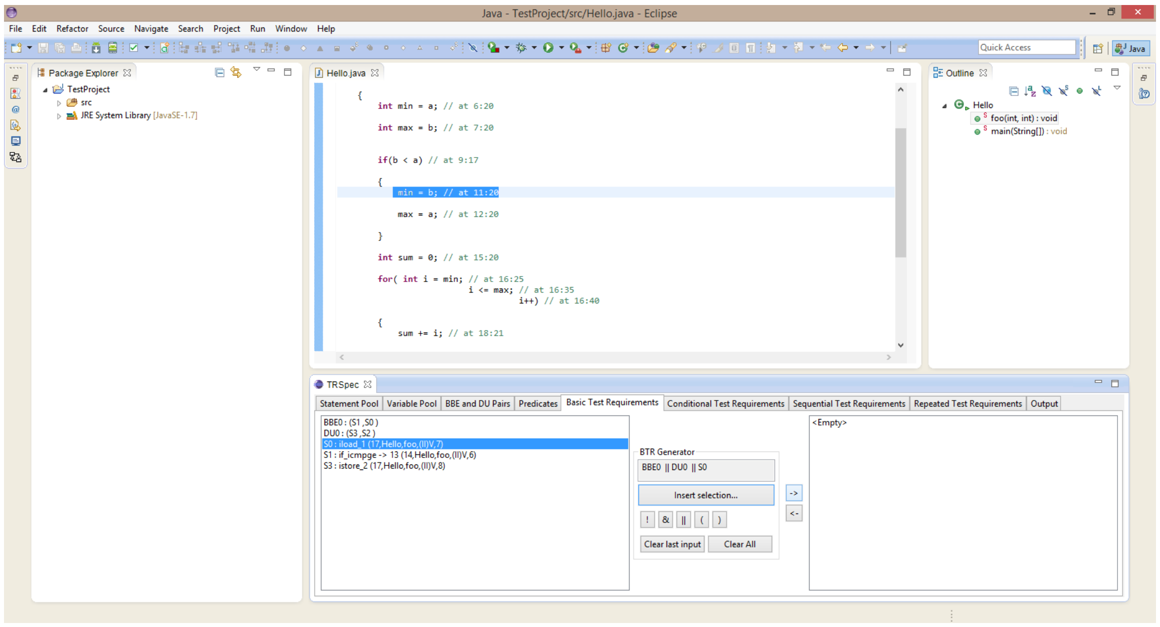
(Var0 == Var1) && (Var1 < 0)

Var1

<Empty>

Basic Test Requirements

1. To construct a basic test requirement (BTR):
    a. You may insert a statement by highlighting it in the list on the left and clicking on the *Insert selection* button.
    b. You may insert a branch or a def-use pair by highlighting it in the same list and clicking on the *Insert selection* button.
    c. You may construct a complex BTRby joining statements, branches, and def-use pairs using the boolean operators&&, ||, and !
2. To clear the last input in the basic test requirement, click on the *Clear Last Input*button. To clear the whole basic test requirement, click on the *Clear All* button.
3. To insert the finished basic test requirement into the pool, click on the → button. To remove abasic test requirement from the pool, highlight it in the list on the right and click on the ←button.



```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                        i <= max; // at 16:35
                                i++) // at 16:40

        {
            sum += i; // at 18:21
```

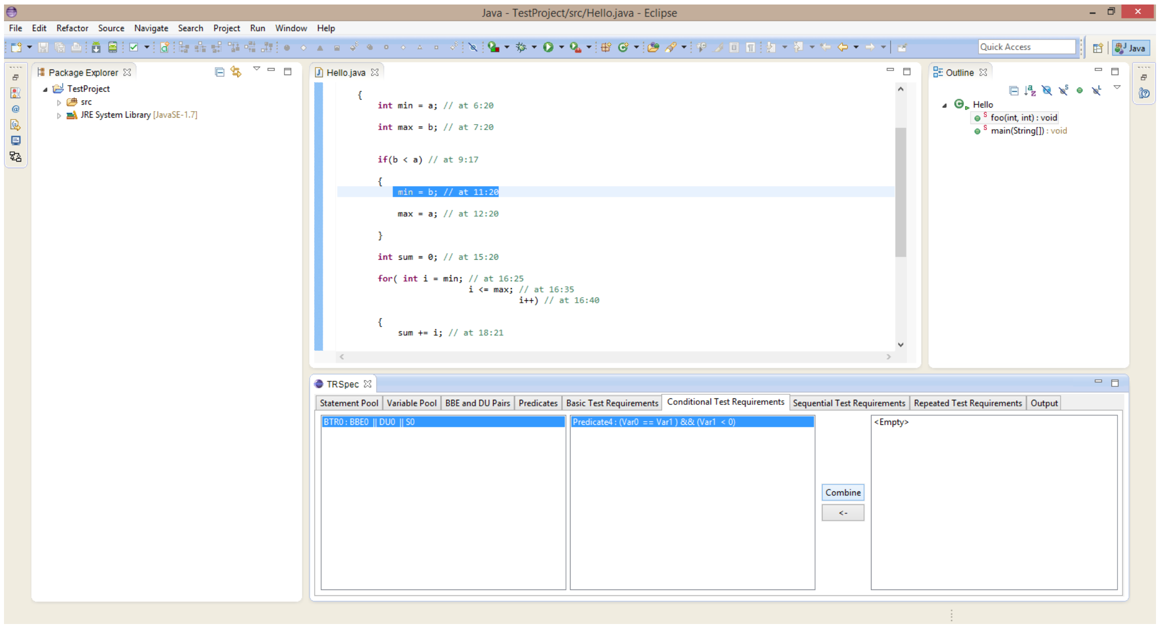
```
BBE0 : (S1 ,S0 )
DU0 : (S3 ,S2 )
S0 : iload_1 (17,Hello,foo,(II)V,7)
S1 : if_icmpge -> 13 (14,Hello,foo,(II)V,6)
S3 : istore_2 (17,Hello,foo,(II)V,8)
```

BBE0 || DU0 || S0

<Empty>

Conditional Test Requirements

1. To construct a conditional test requirement, select one of the (already specified) test requirements from the list on theleft, select a predicate from the middle list, and click on the *Combine* button. This addsit to the pool.
2. To remove a conditional test requirement from the pool, highlight it in the list on the right and click on the ← button.


Java - TestProject/src/Hello.java - Eclipse
File Edit Refactor Source Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
TestProject
src
JRE System Library [JavaSE-1.7]
Hello.java
Outline
Hello
foo(int, int) : void
main(String[]) : void
TRSpec
Statement Pool
Variable Pool
BBE and DU Pairs
Predicates
Basic Test Requirements
Conditional Test Requirements
Sequential Test Requirements
Repeated Test Requirements
Output
Combine
<-

```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                      i <= max; // at 16:35
                            i++) // at 16:40

        {
            sum += i; // at 18:21
```

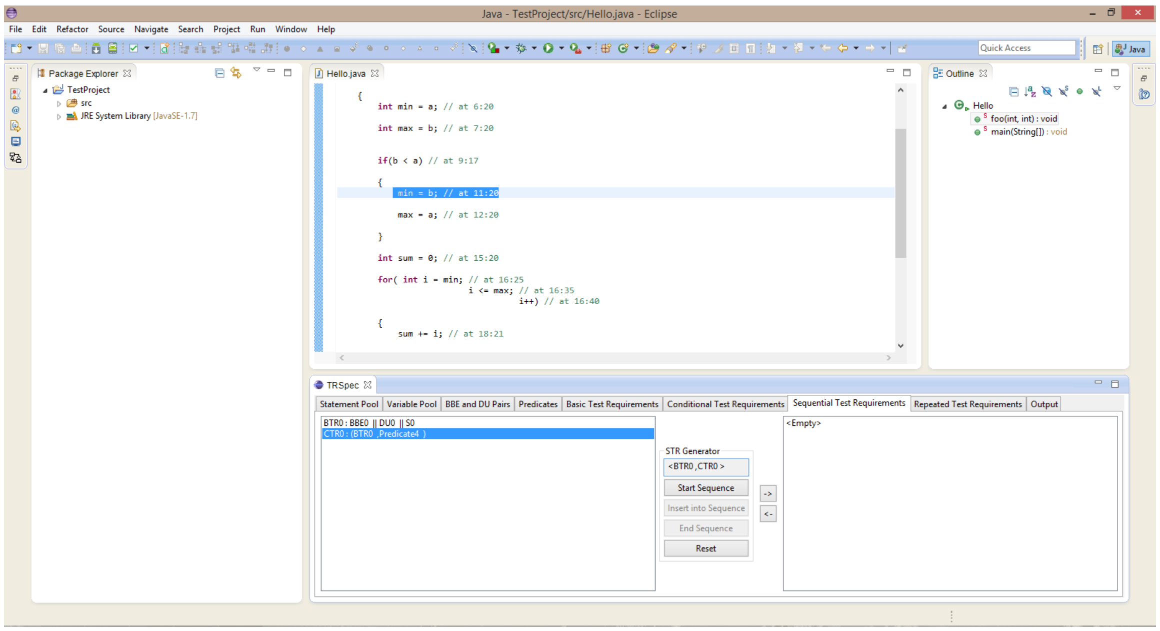
BTR0 : BBE0 || DU0 || S0

Predicate4 : (Var0 == Var1) && (Var1 < 0)

<Empty>

Sequential Test Requirements

1. To construct a sequential test requirement:
    a. Begin the sequence by clicking on the *Start Sequence* button.
    b. Select one of the test requirements from the list on the left and click on the *Insert into Sequence* button. Repeat this for all the elements in the sequence.
    c. When done, click on the *End Sequence* button.
2. You can reset the sequence at any time by clicking on the *Reset* button.
3. To insert the finished sequential test requirement into the pool, click on the ⟶ button. To removea sequential test requirement from the pool, highlight it in the list on the right and click on the ⟵button.



```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                       i <= max; // at 16:35
                              i++) // at 16:40

        {
            sum += i; // at 18:21
```

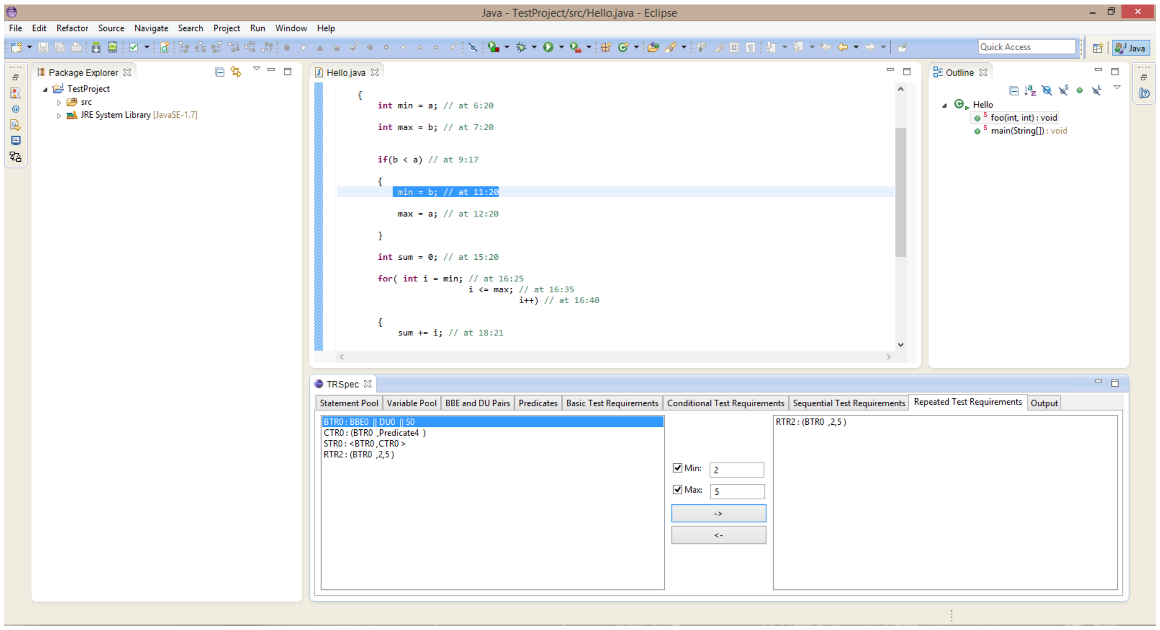
BTR0 : BBE0 || DU0 || S0
CTR0 : (BTR0 ,Predicate4)

<BTR0 ,CTR0 >

<Empty>

Repeated Test Requirements

1. To construct a repeated test requirement:
    a. Selectthe test requirement to be repeated from the list on the left.
    b. To specify a minimum repeat number, mark the corresponding checkbox and type the number in the textbox adjacent to it. If you don't want to specify a minimum repeat number, leave the checkbox unmarked. Similarly, you can specify a maximum repeat number.
2. To insert the finished repeated test requirement into the pool, click on the → button. To remove a repeated test requirement from the pool, highlight it in the list on the right and click on the ← button.


Java - TestProject/src/Hello.java - Eclipse
File Edit Refactor Source Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
TestProject
src
JRE System Library [JavaSE-1.7]
Hello.java
Outline
Hello
foo(int, int) : void
main(String[]) : void
TRSpec
Statement Pool | Variable Pool | BBE and DU Pairs | Predicates | Basic Test Requirements | Conditional Test Requirements | Sequential Test Requirements | Repeated Test Requirements | Output
->
<-

```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                        i <= max; // at 16:35
                                i++) // at 16:40


        {
            sum += i; // at 18:21
```

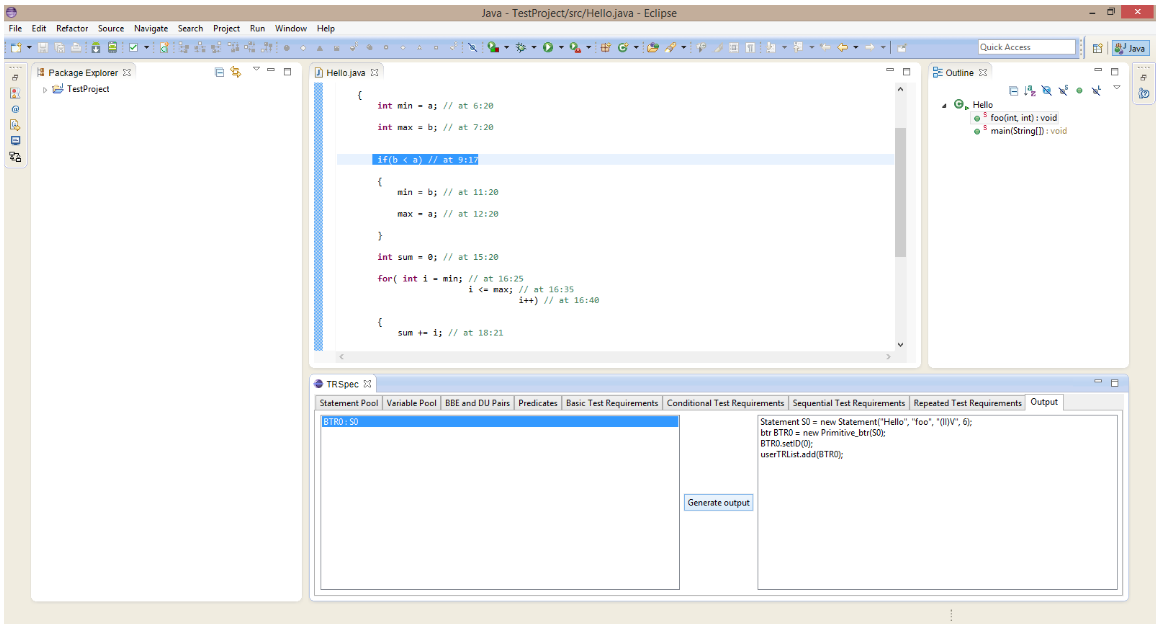
```
BTR0 : BBE0  || DU0  || S0
CTR0 : (BTR0  ,Predicate4  )
STR0 : <BTR0 ,CTR0 >
RTR2 : (BTR0  ,2,5 )
```

☑ Min: 2
☑ Max: 5

```
RTR2 : (BTR0  ,2,5 )
```

Output

To generate compilable code, select the corresponding test requirements from the left pane and click on the *Generate output* button. A pop-out will show asking you to specify the output directory and the generated code will be listed in the right pane.



```
    {
        int min = a; // at 6:20

        int max = b; // at 7:20


        if(b < a) // at 9:17

        {
            min = b; // at 11:20

            max = a; // at 12:20

        }

        int sum = 0; // at 15:20

        for( int i = min; // at 16:25
                        i <= max; // at 16:35
                                i++) // at 16:40

        {
            sum += i; // at 18:21
```

```
Statement S0 = new Statement("Hello", "foo", "(II)V", 6);
btr BTR0 = new Primitive_btr(S0);
BTR0.setID(0);
userTRList.add(BTR0);
```