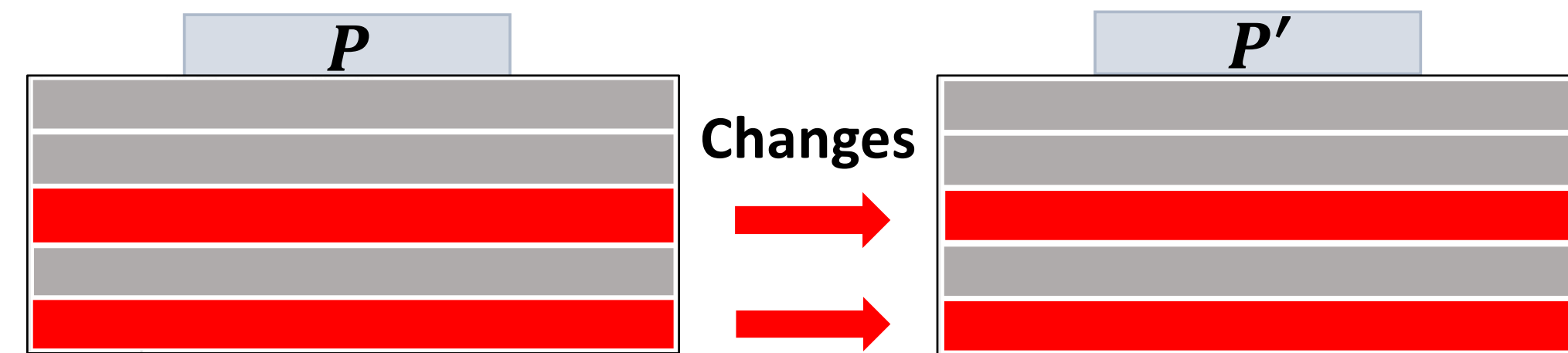


Regression Failures



Unintended different behaviors
between the *base* and the *changed* version of the program

Program Slicing for Debugging

Program slicing computes a subset of program statements that **are related** to a particular variable in a statement [*]

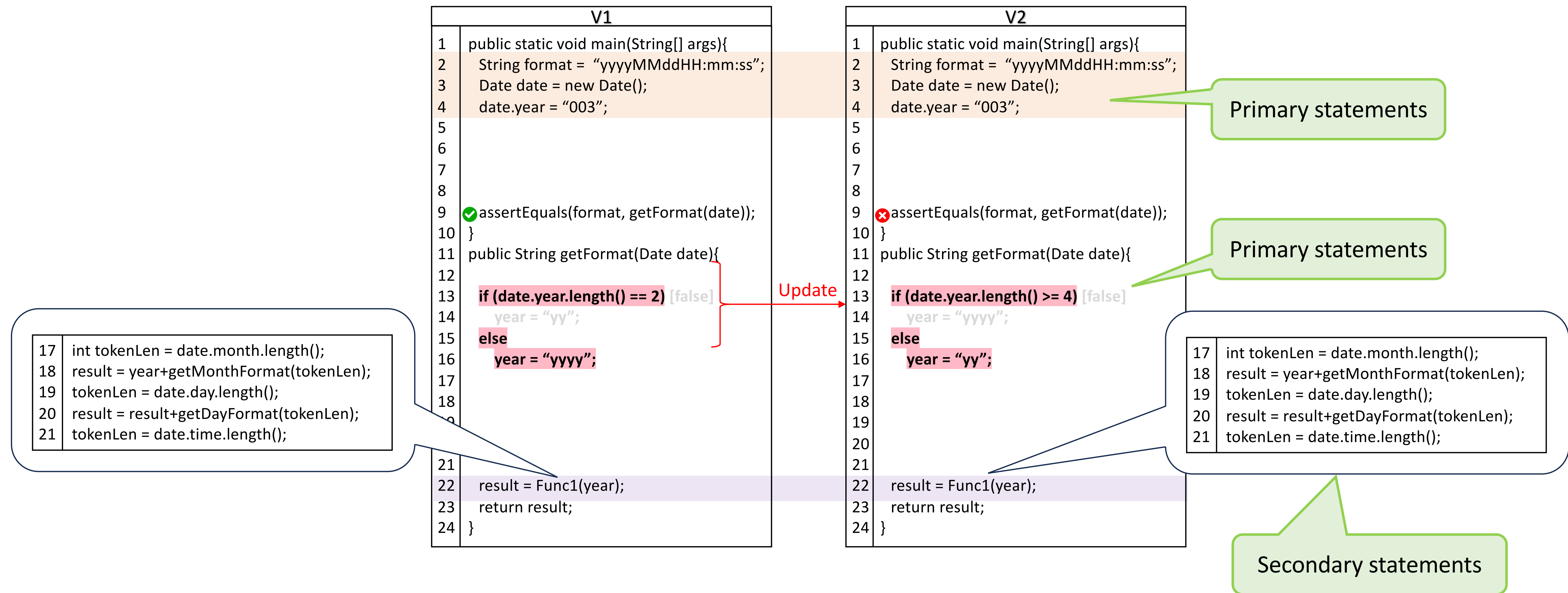
Project	# buggy versions	Statements		
		# trace	# slice	% reduction
Chart	23	5,913	29	95.11
Closure	95	92,663	1,027	98.7
Lang	49	3,091	110	89.57
Math	633	8,801	148	94.49
Mockito	26	3,115	68	97.8
Time	22	13,038	75	99.03
Defects4J Avg.	278	36,017	419	95.78

~35k statements

> 95% reduction of
the code to inspect

~500 statements

Context-Aware Refinement-Based Slicing for Debugging Regression Failures



Focusing on Necessary Information

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() == 2)
17            year = "yy";
18        else
19            year = "yyyy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.time.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String getMonthFormat(int tokenLen){...}
```

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() >= 4)
17            year = "yyyy";
18        else
19            year = "yy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.time.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String getMonthFormat(int tokenLen){...}
```

Highlight the executed changes

Focusing on Necessary Information

```
test master LangNew
```

Lang.java

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12 Assert.assertEquals(format, getFormat(date));
13 }
14 public static String getFormat(Date date){
15     ...
16     if (date.year.length() == 2)
17         year = "yy";
18     else
19         year = "yyyy";
20     int tokenLen = date.month.length();
21     String result = year+getMonthFormat(tokenLen);
22     tokenLen = date.day.length();
23     result = result+getDayFormat(tokenLen);
24     tokenLen = date.time.length();
25     result = result+getTimeFormat(tokenLen);
26     return result;
27 }
28 public static String getMonthFormat(int tokenLen){...}
```

LangNew.java

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12 Assert.assertEquals(format, getFormat(date));
13 }
14 public static String getFormat(Date date){
15     ...
16     if (date.year.length() >= 4)
17         year = "yyyy";
18     else
19         year = "yy";
20     int tokenLen = date.month.length();
21     String result = year+getMonthFormat(tokenLen);
22     tokenLen = date.day.length();
23     result = result+getDayFormat(tokenLen);
24     tokenLen = date.time.length();
25     result = result+getTimeFormat(tokenLen);
26     return result;
27 }
28 public static String getMonthFormat(int tokenLen){...}
```

Keeping Primary statements

Focusing on Necessary Information

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() == 2)
17            year = "yy";
18        else
19            year = "yyyy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.time.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String getMonthFormat(int tokenLen){...}
```

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() >= 4)
17            year = "yyyy";
18        else
19            year = "yy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.time.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String getMonthFormat(int tokenLen){...}
```

Graying out Secondary statements

Focusing on Necessary Information

The image shows a side-by-side comparison of two Java source files in an IDE. The left pane shows `Lang.java` and the right pane shows `LangNew.java`. Both files define a `Lang` class with a `main` method and two static methods: `getFormat` and `getMonthFormat`. In `LangNew.java`, the `if` statement in `getFormat` and the `getMonthFormat` method are highlighted with red boxes, indicating they are irrelevant. Callouts point to these elements with the text "Folding Irrelevant statements" and "Folding Irrelevant methods".

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() == 2)
17            year = "yy";
18        else
19            year = "yyyy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.time.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String getMonthFormat(int tokenLen){...}
```

Folding Irrelevant statements

Folding Irrelevant methods

Focusing on Necessary Information

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() == 2)
17            year = "yy";
18        else
19            year = "yyyy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.time.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String getMonthFormat(int tokenLen){...}
```

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() >= 4)
17            year = "yyyy";
18        else
19            year = "yy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.time.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String getMonthFormat(int tokenLen){...}
```

Unfolding the irrelevant statements

Focusing on Necessary Information

```
test master LangNew
```

Lang.java

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12 Assert.assertEquals(format, getFormat(date));
13 }
14 public static String getFormat(Date date){
15     ...
16     if (date.year.length() == 2)
17         year = "yy";
18     else
19         year = "yyyy";
20     int tokenLen = date.month.length();
21     String result = year+getMonthFormat(tokenLen);
22     tokenLen = date.day.length();
23     result = result+getDayFormat(tokenLen);
24     tokenLen = date.time.length();
25     result = result+getTimeFormat(tokenLen);
26     return result;
27 }
28 public static String getMonthFormat(int tokenLen){...}
```

LangNew.java

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12 Assert.assertEquals(format, getFormat(date));
13 }
14 public static String getFormat(Date date){
15     ...
16     if (date.year.length() >= 4)
17         year = "yyyy";
18     else
19         year = "yy";
20     int tokenLen = date.month.length();
21     String result = year+getMonthFormat(tokenLen);
22     tokenLen = date.day.length();
23     result = result+getDayFormat(tokenLen);
24     tokenLen = date.time.length();
25     result = result+getTimeFormat(tokenLen);
26     return result;
27 }
28 public static String getMonthFormat(int tokenLen){...}
```

Unexecuted statements

Debugging Essential Statements

The screenshot displays an IDE with two tabs: `Lang.java` and `LangNew.java`. Both files contain the following code:

```
package org.example;
import org.junit.Assert;
public class Lang {
    public static void main(String[] args){
        String format = "yyyyMMddHH:mm:ss";
        Date date = new Date();
        date.year = "003";
        ...
        Assert.assertEquals(format, getFormat(date));
    }
    public static String getFormat(Date date){
        ...
        if (date.year.length() == 2)
            year = "yy";
        else
            year = "yyy";
        int tokenLen = date.month.length();
        String result = year+getMonthFormat(tokenLen);
        tokenLen = date.day.length();
        result = result+getDayFormat(tokenLen);
        tokenLen = date.time.length();
        result = result+getTimeFormat(tokenLen);
        return result;
    }
    public static String getMonthFormat(int tokenLen){...}
}
```

The `LangNew.java` version has a modification in the `if` statement condition: `if (date.year.length() >= 4)`. A green callout points to this change with the text "Skipping irrelevant statements".

The `Debug` window shows the `Lang` and `LangNew` frames. The `Console` window displays the output of the `main` method, indicating that the program is running successfully.

Debugging Essential Statements

The screenshot shows an IDE with two tabs: `Lang.java` and `LangNew.java`. Both files contain the same code, but `LangNew.java` has a red dot on line 4, indicating a breakpoint. The code is as follows:

```
package org.example;
import org.junit.Assert;
public class Lang {
    public static void main(String[] args){
        String format = "yyyyMMddHH:mm:ss";
        Date date = new Date();
        date.year = "003";
        ...
        Assert.assertEquals(format, getFormat(date));
    }
    public static String getFormat(Date date){
        ...
        if (date.year.length() == 2)
            year = "yy";
        else
            year = "yyy";
        int tokenLen = date.month.length();
        String result = year+getMonthFormat(tokenLen);
        tokenLen = date.day.length();
        result = result+getDayFormat(tokenLen);
        tokenLen = date.time.length();
        result = result+getTimeFormat(tokenLen);
        return result;
    }
    public static String getMonthFormat(int tokenLen){...}
}
```

The Debug console at the bottom shows the state of variables for both files. The console is split into two panes, one for `Lang` and one for `LangNew`. Both panes show the same state of variables:

- `args` = {String[0]@710} []
- `format` = "yyyyMMddHH:mm:ss"
- `date` = {Date@714}
- `year` = "003"

A green callout bubble points to the console, stating "Displays only important variables".

Debugging Essential Statements

The image shows a side-by-side comparison of two Java source files, `Lang.java` and `LangNew.java`, within an IDE. Both files define a class `Lang` with a `main` method and two helper methods: `getFormat` and `getMonthFormat`.

Lang.java (Left):

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12     Assert.assertEquals(format, getFormat(date));
13 }
14 public static String getFormat(Date date){
15     ...
16     if (date.year.length() == 2)
17         year = "yy";
18     else
19         year = "yyyy";
20     int tokenLen = date.month.length();
21     String result = year+getMonthFormat(tokenLen);
22     tokenLen = date.day.length();
23     result = result+getDayFormat(tokenLen);
24     tokenLen = date.time.length();
25     result = result+getTimeFormat(tokenLen);
26     return result;
27 }
28 public static String getMonthFormat(int tokenLen){...}
```

LangNew.java (Right):

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12     Assert.assertEquals(format, getFormat(date));
13 }
14 public static String getFormat(Date date){
15     ...
16     if (date.year.length() >= 4)
17         year = "yyyy";
18     else
19         year = "yy";
20     int tokenLen = date.month.length();
21     String result = year+getMonthFormat(tokenLen);
22     tokenLen = date.day.length();
23     result = result+getDayFormat(tokenLen);
24     tokenLen = date.time.length();
25     result = result+getTimeFormat(tokenLen);
26     return result;
27 }
28 public static String getMonthFormat(int tokenLen){...}
```

The debugger at the bottom shows the execution state for both files. The console indicates that the `main` method is running. The debugger window shows the current state of the program, with the `date` variable set to `{Date@714}` and the `date.year` variable set to `"003"`.

Selecting Which Statements to Debug on



Debugging Essential Statements

The image shows a side-by-side comparison of two Java files, `Lang.java` and `LangNew.java`, in an IDE. Both files define a `Lang` class with a `main` method and a `getFormat` method. The `main` method creates a `Date` object with year "003" and asserts that the format matches "yyyyMMddHH:mm:ss". The `getFormat` method formats the date based on the length of the year string.

In `Lang.java`, the condition `if (date.year.length() == 2)` is highlighted in red. In `LangNew.java`, the condition `if (date.year.length() >= 4)` is highlighted in red. An orange arrow points from the `else` block of `LangNew.java` to the `return result;` statement.

Below the code, the debugger console shows the state of variables `date`, `year`, and `result` for both files. In `Lang.java`, `year` is "yyyy". In `LangNew.java`, `year` is "yy". A green callout box labeled "Different values" points to these two states.

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() == 2)
17            year = "yy";
18        else
19            year = "yyyy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.time.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String getMonthFormat(int tokenLen){...}
```

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() >= 4)
17            year = "yyyy";
18        else
19            year = "yy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.time.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String getMonthFormat(int tokenLen){...}
```

Debug Console (Lang):

```
"main"@1 i...n": RUNNING
main:19 , Lang (org.example)
> date = {Date@714}
> year = "yyyy" [Diff]
> result = "yyyyMMddHH:mm:ss" :func(year)
```

Debug Console (LangNew):

```
"main"@1 i...n": RUNNING
main:19 , LangNew (org.example)
> date = {Date@714}
> year = "yy" [Diff]
> result = "yyMMddHH:mm:ss" :func(year)
```

Different values

Side-by-Side Debugging

test master

LangNew

Lang.java

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12 Assert.assertEquals(format, getFormat(date));
13 }
14 public static String getFormat(Date date){
15     ...
16     if (date.year.length() == 2)
17         year = "yy";
18     else
19         year = "yyyy";
20     int tokenLen = date.month.length();
21     String result = year+getMonthFormat(tokenLen);
22     tokenLen = date.day.length();
23     result = result+getDayFormat(tokenLen);
24     tokenLen = date.time.length();
25     result = result+getTimeFormat(tokenLen);
26     return result;
27 }
28 public static String getMonthFormat(int tokenLen){...}
```

LangNew.java

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12 Assert.assertEquals(format, getFormat(date));
13 }
14 public static String getFormat(Date date){
15     ...
16     if (date.year.length() >= 4)
17         year = "yyyy";
18     else
19         year = "yy";
20     int tokenLen = date.month.length();
21     String result = year+getMonthFormat(tokenLen);
22     tokenLen = date.day.length();
23     result = result+getDayFormat(tokenLen);
24     tokenLen = date.time.length();
25     result = result+getTimeFormat(tokenLen);
26     return result;
27 }
28 public static String getMonthFormat(int tokenLen){...}
```

Debug Lang

Console

Debugger

"main"@1 i...n": RUNNING

main:19 , Lang (org.example)

Evaluate expression (⌘) or add a watch (⇧⌘)

date = {Date@714}

year = "yyyy" [Diff]

result = "yyyyMMddHH:mm:ss" :func(year)

Debug LangNew

Console

Debugger

"main"@1 i...n": RUNNING

main:19, LangNew (org.example)

Evaluate expression (⌘) or add a watch (⇧⌘)

date = {Date@714}

year = "yy" [Diff]

result = "yyMMddHH:mm:ss" :func(year)

Sync the debugging

Textual Explanations

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() == 2)
17            year = "yy";
18        else
19            year = "yyyy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.hour.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String ...

```

The variable *result* is assigned in line 21 based on the value of *year* (line 19) and the output of the method *getMonthFormat*. It is further reassigned in lines 23 and 25, based on its previous value and the output of the methods *getDayFormat* (line 23) and *getTimeFormat* (line 25) applied on *tokenLen*.

```
1 package org.example;
2 import org.junit.Assert;
3 public class Lang {
4     public static void main(String[] args){
5         String format = "yyyyMMddHH:mm:ss";
6         Date date = new Date();
7         date.year = "003";
8         ...
12        Assert.assertEquals(format, getFormat(date));
13    }
14    public static String getFormat(Date date){
15        ...
16        if (date.year.length() >= 4)
17            year = "yyyy";
18        else
19            year = "yy";
20        int tokenLen = date.month.length();
21        String result = year+getMonthFormat(tokenLen);
22        tokenLen = date.day.length();
23        result = result+getDayFormat(tokenLen);
24        tokenLen = date.hour.length();
25        result = result+getTimeFormat(tokenLen);
26        return result;
27    }
28    public static String ...

```

The variable *result* is assigned in line 21 based on the value of *year* (line 19) and the output of the method *getMonthFormat*. It is further reassigned in lines 23 and 25, based on its previous value and the output of the methods *getDayFormat* (line 23) and *getTimeFormat* (line 25) applied on *tokenLen*.