Introduction to Software Testing

Chapter 3
Logic Coverage for Source Code
Logic Coverage for Specification

Paul Ammann & Jeff Offutt

Updated by Sunae Shin

Logic Expressions from Source

- Predicates are derived from decision statements in programs

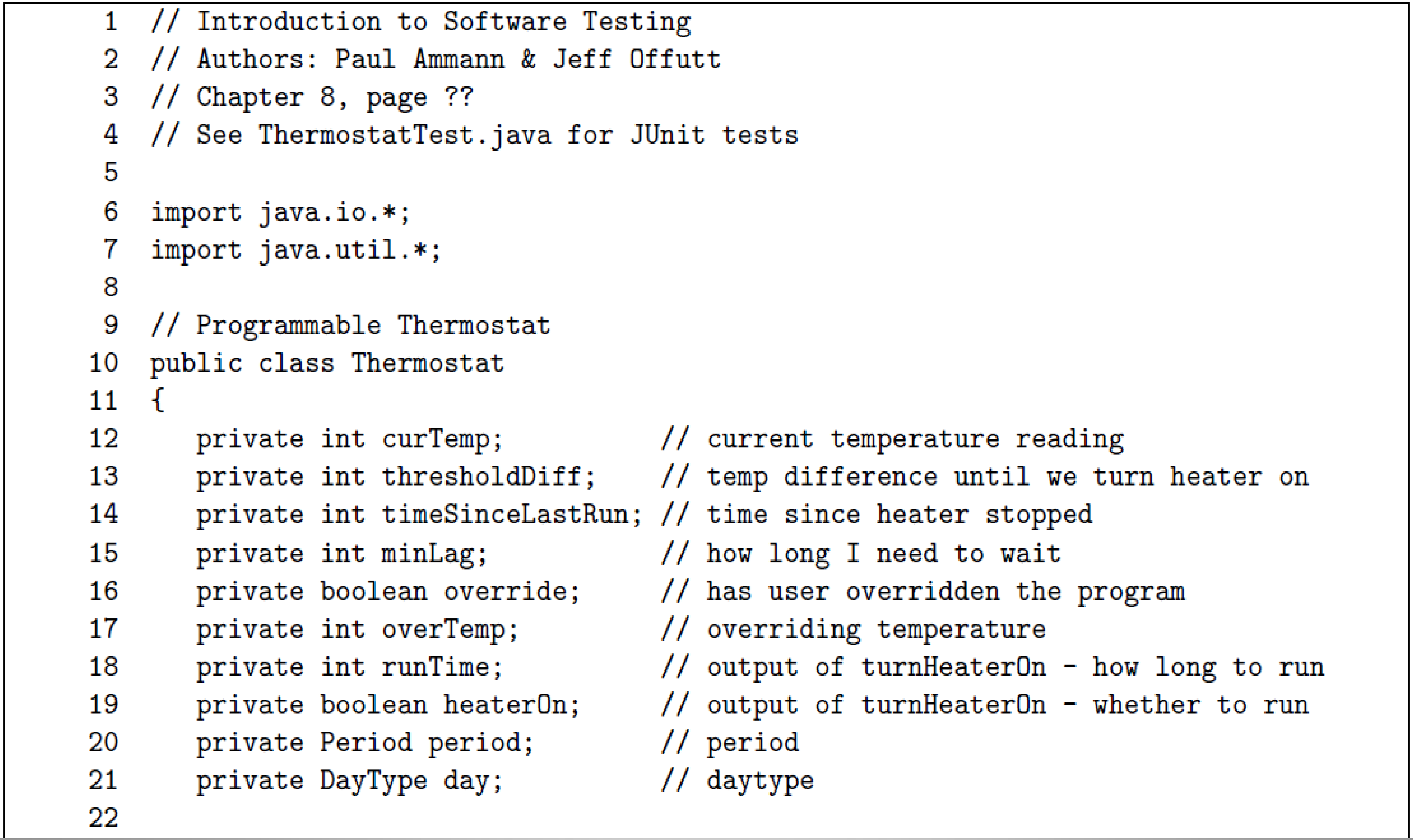
- In programs, most predicates have less than three clauses
  - Wise programmers actively strive to keep predicates simple

- When a predicate only has one clause, COC, ACC, and CC all collapse to predicate coverage (PC)

Logic Expressions from Source

- Applying logic criteria to program source is hard because of reachability and controllability:
  - *Reachability* : Before applying the criteria on a predicate at a particular statement, we have to get to that statement
  - *Controllability* : We have to find input values that indirectly assign values to the variables in the predicates
  - Variables in the predicates that are not inputs to the program are called *internal variables*
- These issues are illustrated through an example in the following slides …

Thermostat (*pg 1 of 2*)



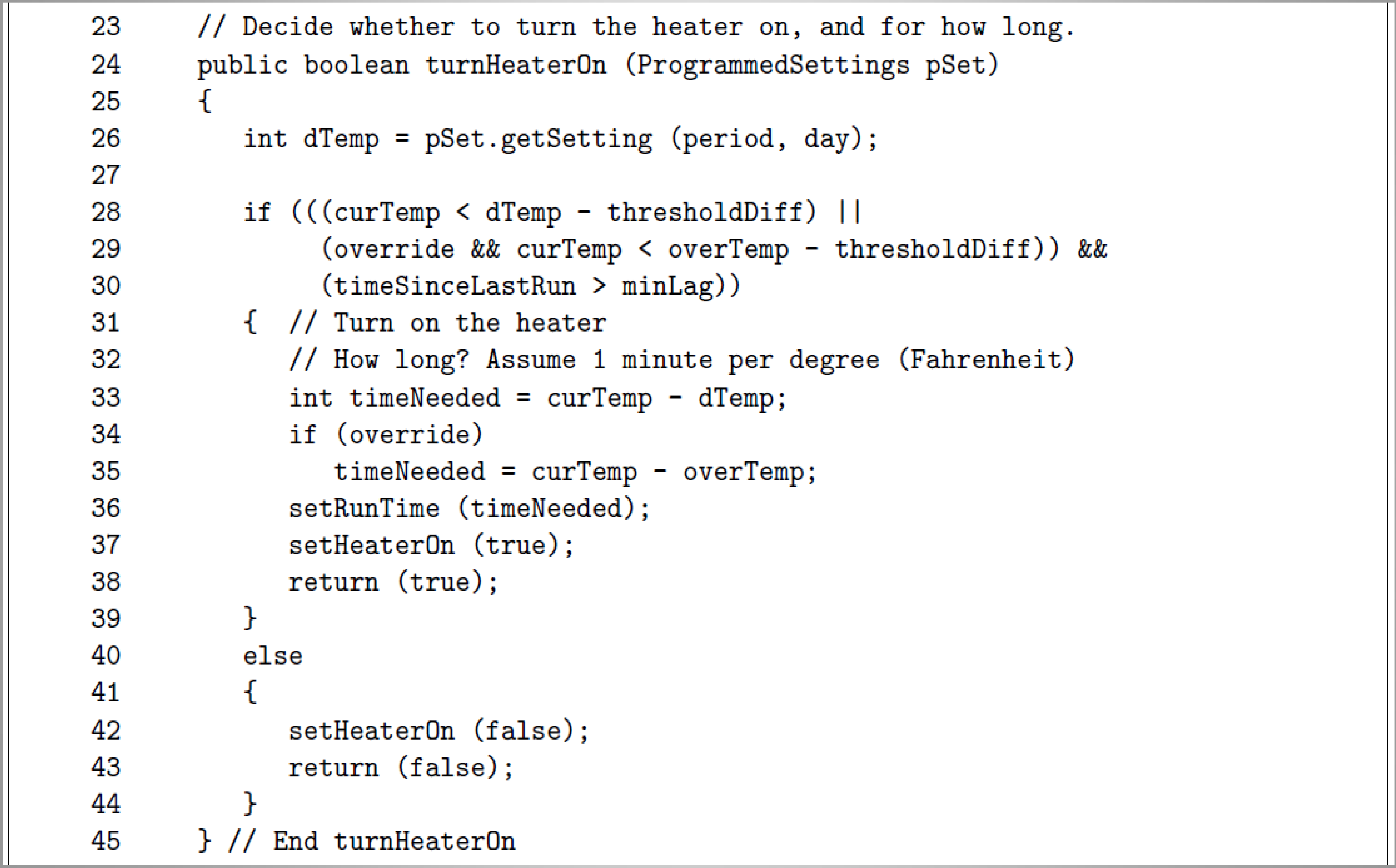
```java
// Introduction to Software Testing
// Authors: Paul Ammann & Jeff Offutt
// Chapter 8, page ??
// See ThermostatTest.java for JUnit tests

import java.io.*;
import java.util.*;

// Programmable Thermostat
public class Thermostat
{
   private int curTemp;           // current temperature reading
   private int thresholdDiff;     // temp difference until we turn heater on
   private int timeSinceLastRun;  // time since heater stopped
   private int minLag;            // how long I need to wait
   private boolean override;      // has user overridden the program
   private int overTemp;          // overriding temperature
   private int runTime;           // output of turnHeaterOn - how long to run
   private boolean heaterOn;      // output of turnHeaterOn - whether to run
   private Period period;         // period
   private DayType day;           // daytype

```

Thermostat (*pg 2 of 2*)



```
// Decide whether to turn the heater on, and for how long.
public boolean turnHeaterOn (ProgrammedSettings pSet)
{
   int dTemp = pSet.getSetting (period, day);

   if (((curTemp < dTemp - thresholdDiff) ||
        (override && curTemp < overTemp - thresholdDiff)) &&
        (timeSinceLastRun > minLag))
   {  // Turn on the heater
      // How long? Assume 1 minute per degree (Fahrenheit)
      int timeNeeded = curTemp - dTemp;
      if (override)
         timeNeeded = curTemp - overTemp;
      setRunTime (timeNeeded);
      setHeaterOn (true);
      return (true);
   }
   else
   {
      setHeaterOn (false);
      return (false);
   }
} // End turnHeaterOn
```

Two Thermostat Predicates

28-30 : (((curTemp < dTemp - thresholdDiff) ||
(override && curTemp < overTemp - thresholdDiff)) &&
timeSinceLastRun > minLag)

34 : (override)

Simplify

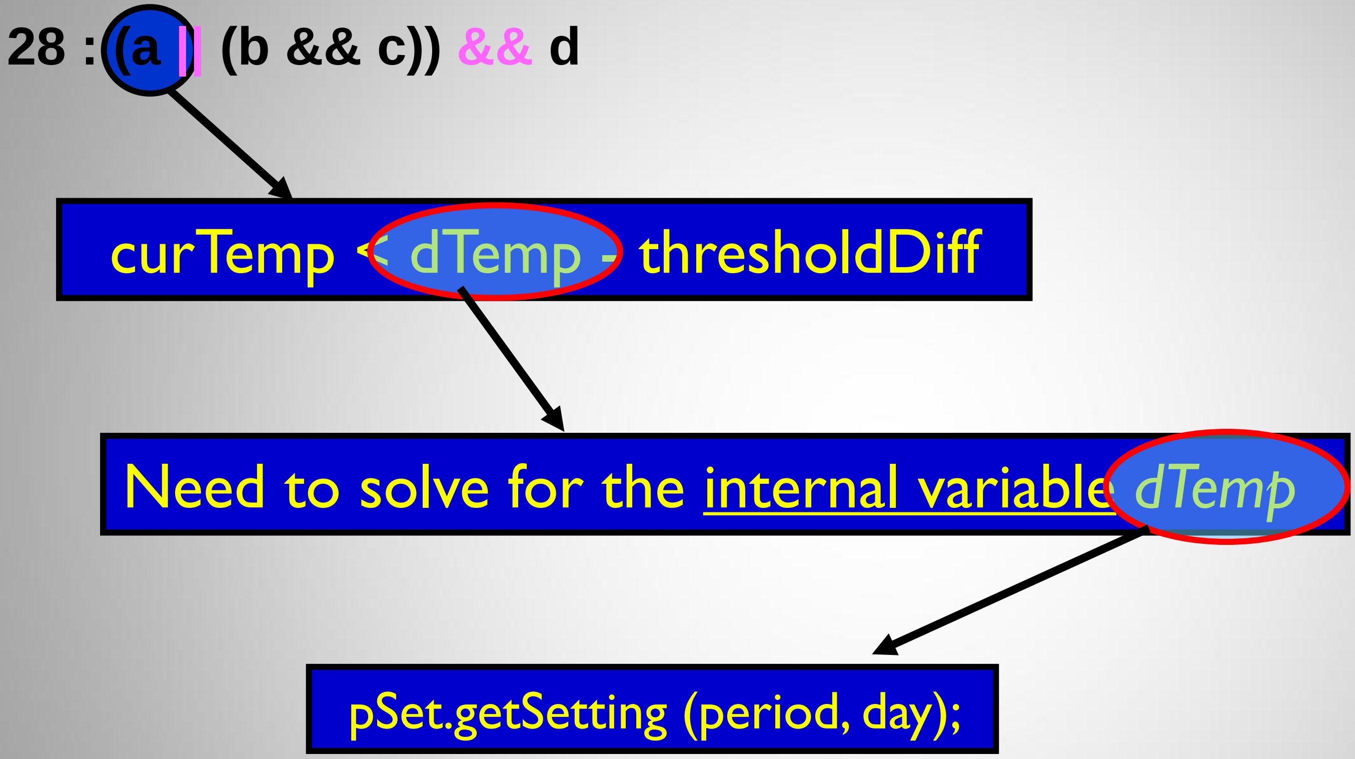
a : curTemp < dTemp - thresholdDiff
b : override
c : curTemp < overTemp - thresholdDiff
d : timeSinceLastRun > minLag

28-30 : (a || (b && c)) && d

34 : b

Reachability for Thermostat Predicates

34 : True



28 : (a || (b && c)) && d

curTemp < dTemp - thresholdDiff

Need to solve for the internal variable *dTemp*

pSet.getSetting (period, day);

```
setSetting (Period.MORNING, DayType.WEEKDAY, 69);
setPeriod (Period.MORNING);
setDay (DayType.WEEKDAY);
```

Predicate Coverage (*true*)

(a || (b && c)) && d

a : true　　b : true
c : true　　d : true

a: curTemp < dTemp – thresholdDiff : true
b: Override : true
c: curTemp < overTemp – thresholdDiff : true
d: timeSinceLastRun > (minLag) : true

```
thermo = new Thermostat(); // Needed object
settings = new ProgrammedSettings(); // Needed object
settings.setSetting (Period.MORNING, DayType.WEEKDAY, 69); // dTemp
thermo.setPeriod (Period.MORNING); // dTemp
thermo.setDay (DayType.WEEKDAY); // dTemp
thermo.setCurrentTemp (63); // clause a
thermo.setThresholdDiff (5);  // clause a
thermo.setOverride (true); // clause b
thermo.setOverTemp (70); // clause c
thermo.setMinLag (10); // clause d
thermo.setTimeSinceLastRun (12); // clause d
assertTrue (thermo.turnHeaterOn (settings));  // Run test
```

Predicate Coverage (*false*)

(a || (b && c)) && d

a : false b : false
c : false d : false

a: curTemp < dTemp – thresholdDiff : false
b: Override : false
c: curTemp < overTemp – thresholdDiff : false
d: timeSinceLastRun > (minLag) : false

```
thermo = new Thermostat();  // Needed object
settings = new ProgrammedSettings();  // Needed object
settings.setSetting (Period.MORNING, DayType.WEEKDAY, 69);  // dTemp
thermo.setPeriod (Period.MORNING);  // dTemp
thermo.setDay (DayType.WEEKDAY);  // dTemp
thermo.setCurrentTemp (66);  // clause a
thermo.setThresholdDiff (5);   // clause a
thermo.setOverride (false);  // clause b
thermo.setOverTemp (70);  // clause c
thermo.setMinLag (10);  // clause d
thermo.setTimeSinceLastRun (8);  // clause d
assertTrue (thermo.turnHeaterOn (settings));   // Run test
```

Correlated Active Clause Coverage

(1 of 6)

$$P_a = ((a \,||\, (b \,\&\&\, c)) \,\&\&\, d) \oplus ((a \,||\, (b \,\&\&\, c)) \,\&\&\, d)$$

$$((T \,||\, (b \,\&\&\, c)) \,\&\&\, d) \oplus ((F \,||\, (b \,\&\&\, c)) \,\&\&\, d)$$

$$(T \,\&\&\, d) \oplus ((b \,\&\&\, c) \,\&\&\, d)$$

$$d \oplus ((b \,\&\&\, c) \,\&\&\, d)$$

$$T \oplus ((b \,\&\&\, c) \,\&\&\, T)$$

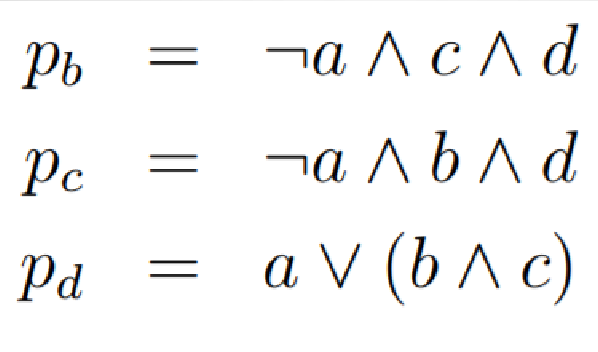
$$!(b \,\&\&\, c) \,\&\&\, d$$

$$(\,!b \,||\, !c\,) \,\&\&\, d$$

Clause *a* determines the value of the predicate exactly when *d* is true, and either *b* or *c* is false.

Correlated Active Clause Coverage

Similar computations for clauses b, c, and d yield:



$$
\begin{aligned}
p_b &= \neg a \wedge c \wedge d \\
p_c &= \neg a \wedge b \wedge d \\
p_d &= a \vee (b \wedge c)
\end{aligned}
$$

Correlated Active Clause Coverage

(2 of 6)

(a || (b && c)) && d

	a	b	c	d
P_a :	T	t	f	t
	F	t	f	t
P_b :	f	T	t	t
	f	F	t	t
P_c :	~~f~~	~~t~~	~~T~~	~~t~~
	~~f~~	~~t~~	~~F~~	~~t~~
P_d :	t	t	t	T
	t	t	t	F

duplicates

Six tests needed for CACC on Thermostat

Correlated Active Clause Coverage (3 of 6)

	curTemp	dTemp	thresholdDiff
a=t : curTemp < dTemp - thresholdDiff	63	69	5
a=f : !(curTemp < dTemp - thresholdDiff)	66	69	5

dTemp:

```
   settings.setSettings (Period.MORNING, DayType.WEEKDAY, 69)
   thermo.setPeriod (Period.MORNING);
   thermo.setDay (Daytype.WEEKDAY);
```

	Override
b=t : Override	T
b=f : !Override	F

These values then need to be placed into calls to turnHeaterOn() to satisfy the 6 tests for CACC

	curTemp	overTemp	thresholdDiff
c=t : curTemp < overTemp - thresholdDiff	63	72	5
c=f : !(curTemp < overTemp - thresholdDiff)	66	67	5

	timeSinceLastRun	minLag
d=t : timeSinceLastRun > minLag	12	10
d=f : !(timeSinceLastRun > minLag)	8	10

Correlated Active Clause Coverage

(4 of 6)

dTemp = 69 (period = MORNING, daytype = WEEKDAY)

1. T t f t

```
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (67); // c is false
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

2. F t f t

```
thermo.setCurrentTemp (66); // a is false
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (67); // c is false
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

Correlated Active Clause Coverage

(5 of 6)

dTemp = 69 (period = MORNING, daytype = WEEKDAY)

3. f T t t

```
   thermo.setCurrentTemp (66); // a is false
   thermo.setThresholdDiff (5);
   thermo.setOverride (true);
   thermo.setOverTemp (72); // to make c true
   thermo.setMinLag (10);
   thermo.setTimeSinceLastRun (12);
```

4. F f T t

```
   thermo.setCurrentTemp (66); // a is false
   thermo.setThresholdDiff (5);
   thermo.setOverride (false); // b is false
   thermo.setOverTemp (72);
   thermo.setMinLag (10);
   thermo.setTimeSinceLastRun (12);
```

Correlated Active Clause Coverage

(6 of 6)

dTemp = 69 (period = MORNING, daytype = WEEKDAY)

5. t t t T

```
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

6. t t t F

```
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (8); // d is false
```

Summary : Logic Coverage for Source Code

- Predicates appear in decision statements (if, while, for, etc.)
- Most predicates have less than four clauses
  - But some programs have a few predicates with many clauses
- The hard part of applying logic criteria to source is usually resolving the internal variables
  - Sometimes setting variables requires calling other methods

Specification Logic

Specifications in Software

- Specifications can be formal or informal
  - Formal specs are usually expressed mathematically
  - Informal specs are usually expressed in *natural language*

- Lots of formal languages and informal styles are available

- Most specification languages include explicit logical expressions, so it is very easy to apply logic coverage criteria

- Implicit logical expressions in natural-language specifications should be re-written as explicit logical expressions as part of test design
  - You will often find mistakes

- One of the most common is preconditions …

Preconditions

- Programmers often include preconditions for their methods
- The preconditions are often expressed in comments in method headers
- Preconditions can be in javadoc, “requires”, “pre”, …

Example – Saving addresses

// name must not be empty

// state must be valid

// zip must be 5 numeric digits

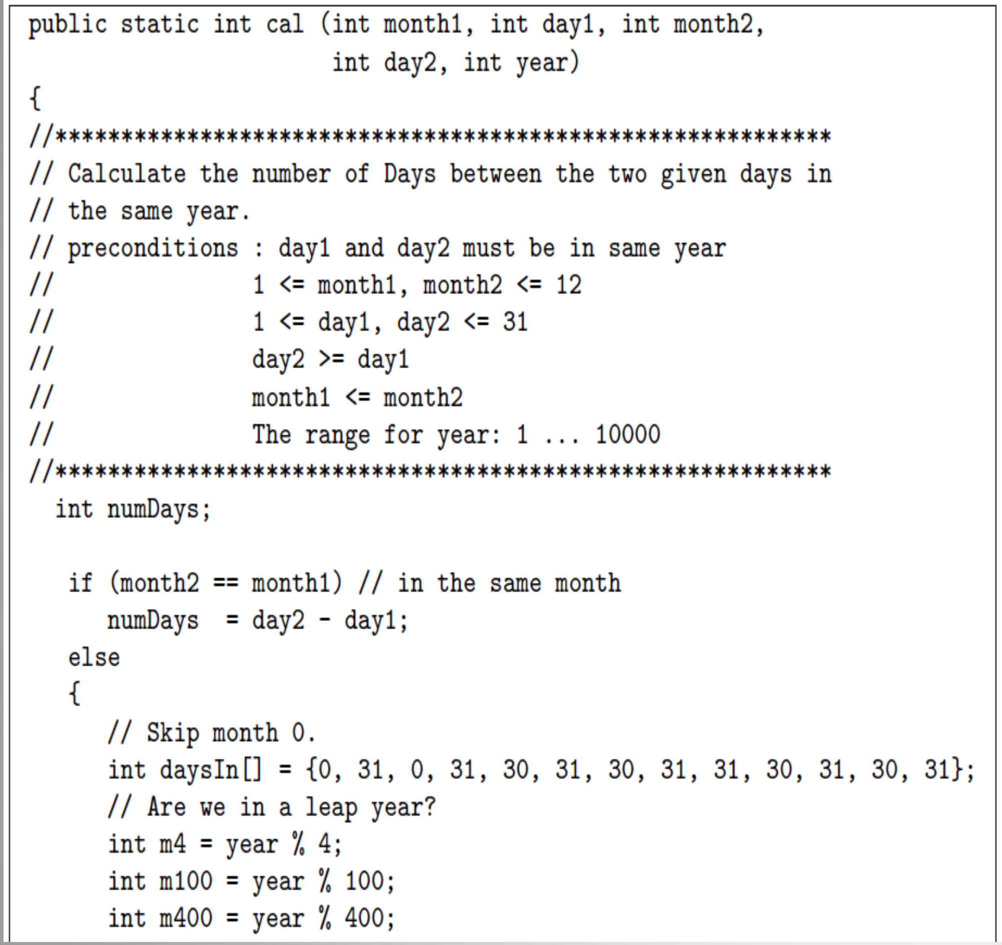
// street must not be empty

// city must not be empty

Rewriting to logical expression

name != “” ∧ state in stateList ∧ zip >= 00000 ∧ zip <= 99999 ∧ street != “” ∧ city != “”

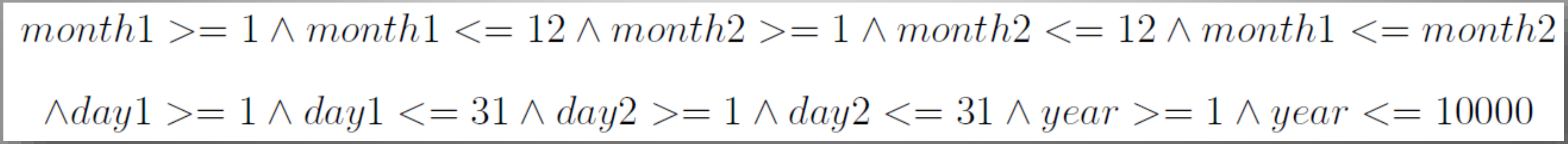
Preconditions - example



```
public static int cal (int month1, int day1, int month2,
                       int day2, int year)
{
//***********************************************************
// Calculate the number of Days between the two given days in
// the same year.
// preconditions : day1 and day2 must be in same year
//                 1 <= month1, month2 <= 12
//                 1 <= day1, day2 <= 31
//                 day2 >= day1
//                 month1 <= month2
//                 The range for year: 1 ... 10000
//***********************************************************
  int numDays;

   if (month2 == month1) // in the same month
      numDays  = day2 - day1;
   else
   {
      // Skip month 0.
      int daysIn[] = {0, 31, 0, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
      // Are we in a leap year?
      int m4 = year % 4;
      int m100 = year % 100;
      int m400 = year % 400;
```

- The method lists explicit preconditions in natural language.
- These can be translated into predicate form as follows:



$$month1 >= 1 \wedge month1 <= 12 \wedge month2 >= 1 \wedge month2 <= 12 \wedge month1 <= month2$$

$$\wedge day1 >= 1 \wedge day1 <= 31 \wedge day2 >= 1 \wedge day2 <= 31 \wedge year >= 1 \wedge year <= 10000$$

Preconditions – example (contd)

$$month1 >= 1 \wedge month1 <= 12 \wedge month2 >= 1 \wedge month2 <= 12 \wedge month1 <= month2$$

$$\wedge day1 >= 1 \wedge day1 <= 31 \wedge day2 >= 1 \wedge day2 <= 31 \wedge year >= 1 \wedge year <= 10000$$

- This predicate has a very simple structure
  - It has eleven clauses
  - but the only logical operator is “and”

- Satisfying **predicate coverage**
  - all clauses need to be true for the true case and at least one clause needs to be false for the false case
  - So (*month1 = 4, month2 = 4, day1 = 12, day2 = 30, year = 1961)* satisfies the true case, and
  - the false case is satisfied by violating the clause month1 <= month2, with *(month1 = 6, month2 = 4, day1 = 12, day2 = 30, year = 1961)*

- **Clause coverage** requires all clauses to be true and false

Summary : Logic Coverage for Specs

- Logical specifications can come from lots of places :
  - Preconditions
  - Java asserts
  - Contracts (in design-by-contract development)
  - Formal languages

- Logical specifications can describe behavior at many levels :
  - Methods and classes (unit and module testing)
  - Connections among classes and components
  - System-level behavior