Control Flow Part I

CS 18000

Sunil Prabhakar

Department of Computer Science

Purdue University

Problem

- *Write a program that tells a patient if their total cholesterol measure is too high or not.*
  - *The measure is an integer and is too high if it exceeds 239.*
- *Your program should read in the measure and output an appropriate evaluation.*

Choices

- Clearly, in order to solve this problem, we need to be able to choose which of the alternative messages to print.
- All programming languages provide this ability to choose: selection statements.
- Java provides **if-else**, and **switch** selection statements.
- This week we will study **if-else** statements

Flow of control

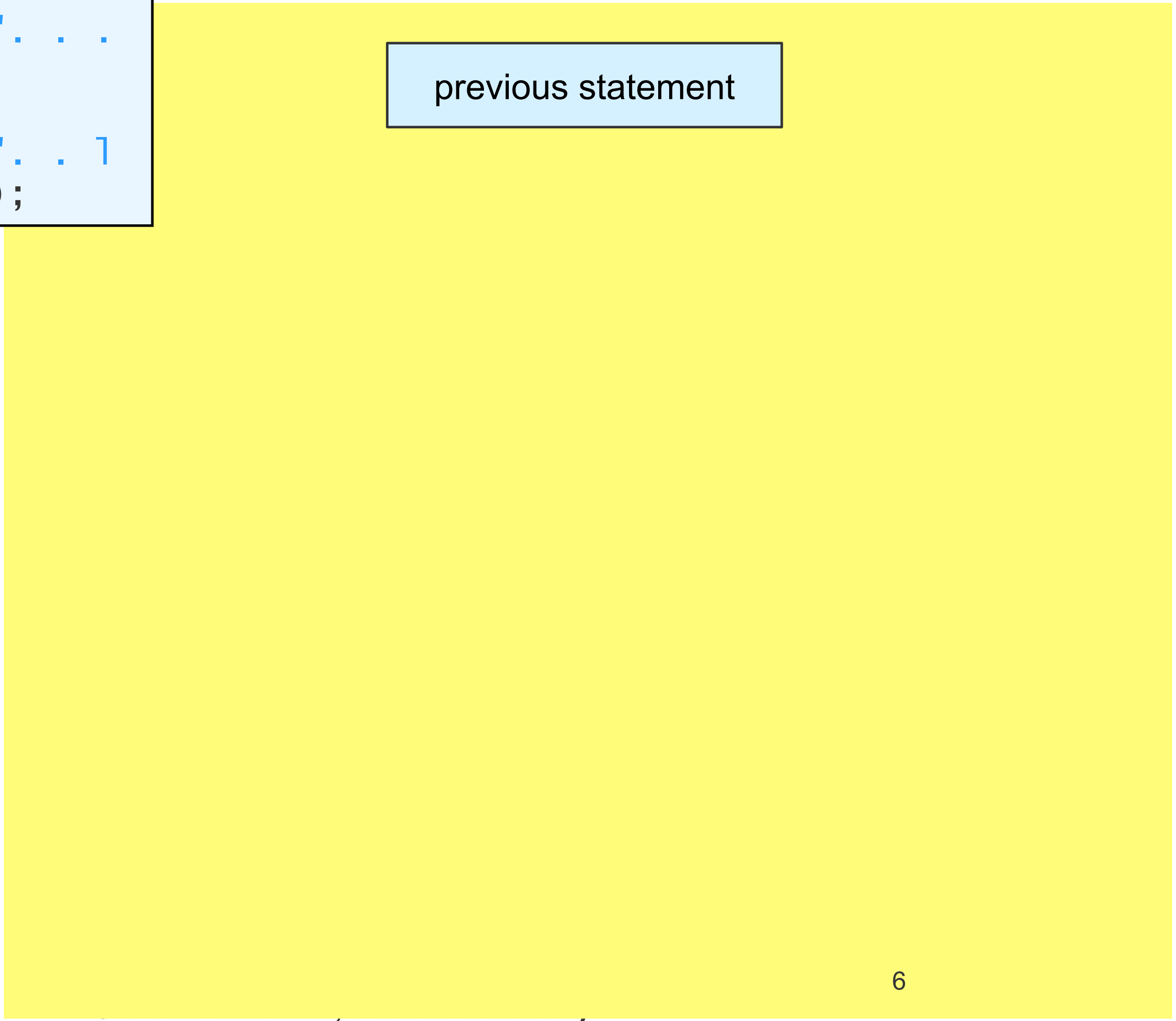
- Once a statement is executed, the next statement of the program is executed.
- Calling a method transfers the control to the statements in the method.
  - Once the method returns, control returns to statement that made the call.
- Changing this flow of control is achieved using **if-else** and **do-while** etc. statements.
- These are called control flow statements.

Solution

```
public class CholesterolCheck {
   public static void main(String[] args){

      int chLevel;

      chLevel = Integer.parseInt(JOptionPane.showInputDialog(
                   null, "Enter your cholesterol measure"));


      if(chLevel > 239)
         System.out.print("Your cholesterol level is too high.");
      else
         System.out.print("Your cholesterol level is not too
         high.");


   }
}
```

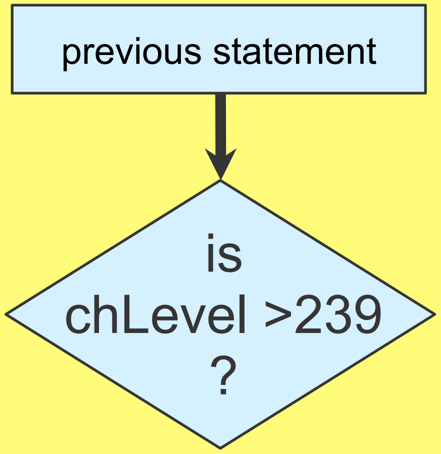
if-else Control Flow

```
if (chLevel > 239)
         System.out.print(". . .
         is too high.");
else
         System.out.print(". . l
         is not too high.");
```



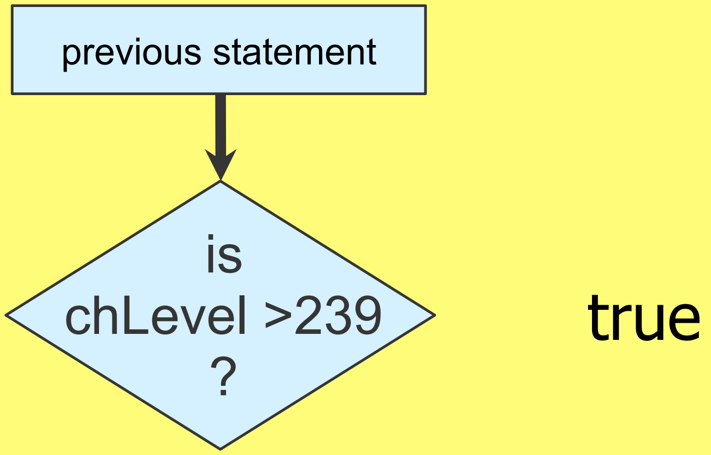
if-else Control Flow

```
if (chLevel > 239)
          System.out.print(". . .
          is too high.");
else
          System.out.print(". . l
          is not too high.");
```



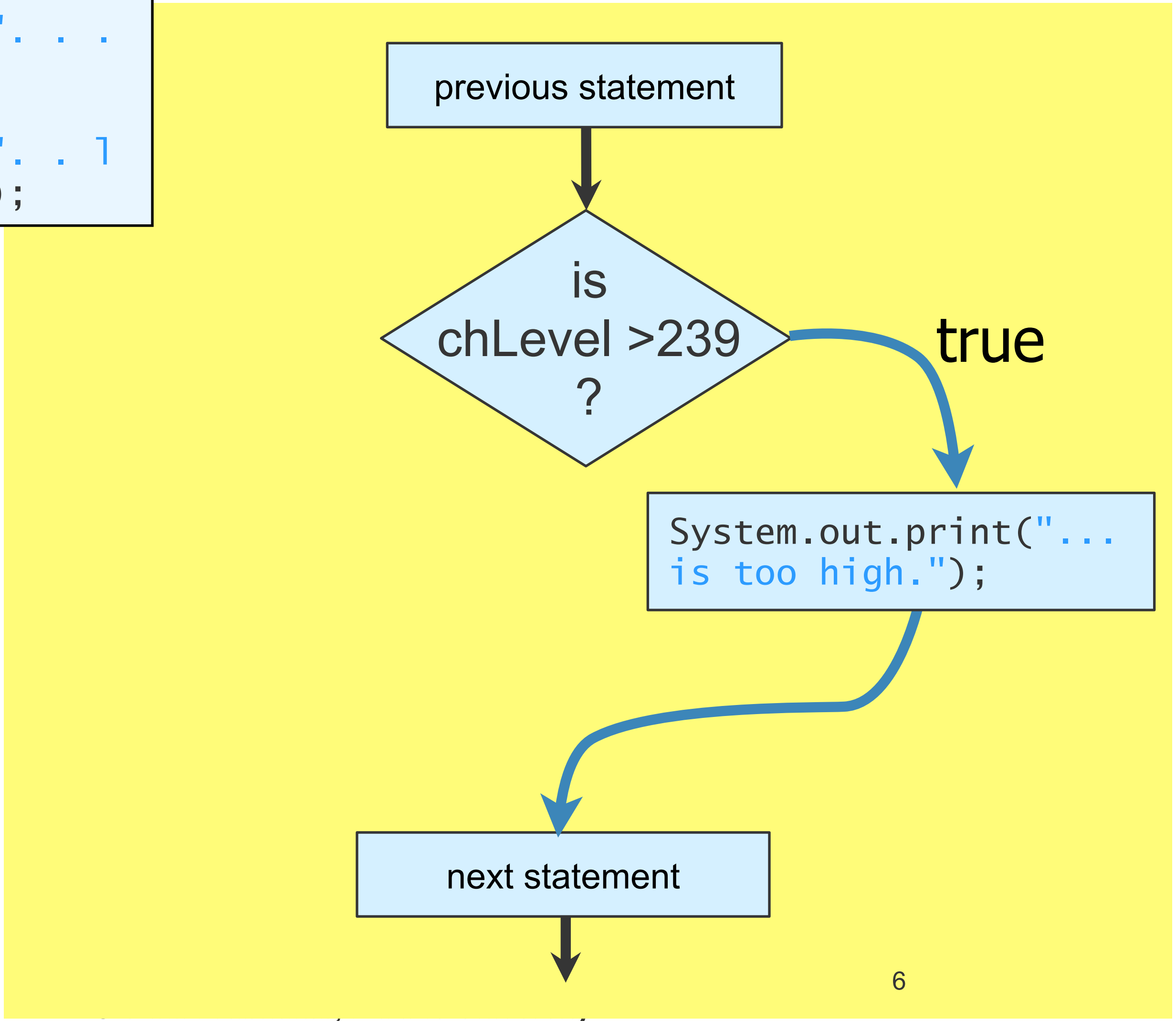
if-else Control Flow

```
if (chLevel > 239)
        System.out.print(". . .
        is too high.");
else
        System.out.print(". . l
        is not too high.");
```



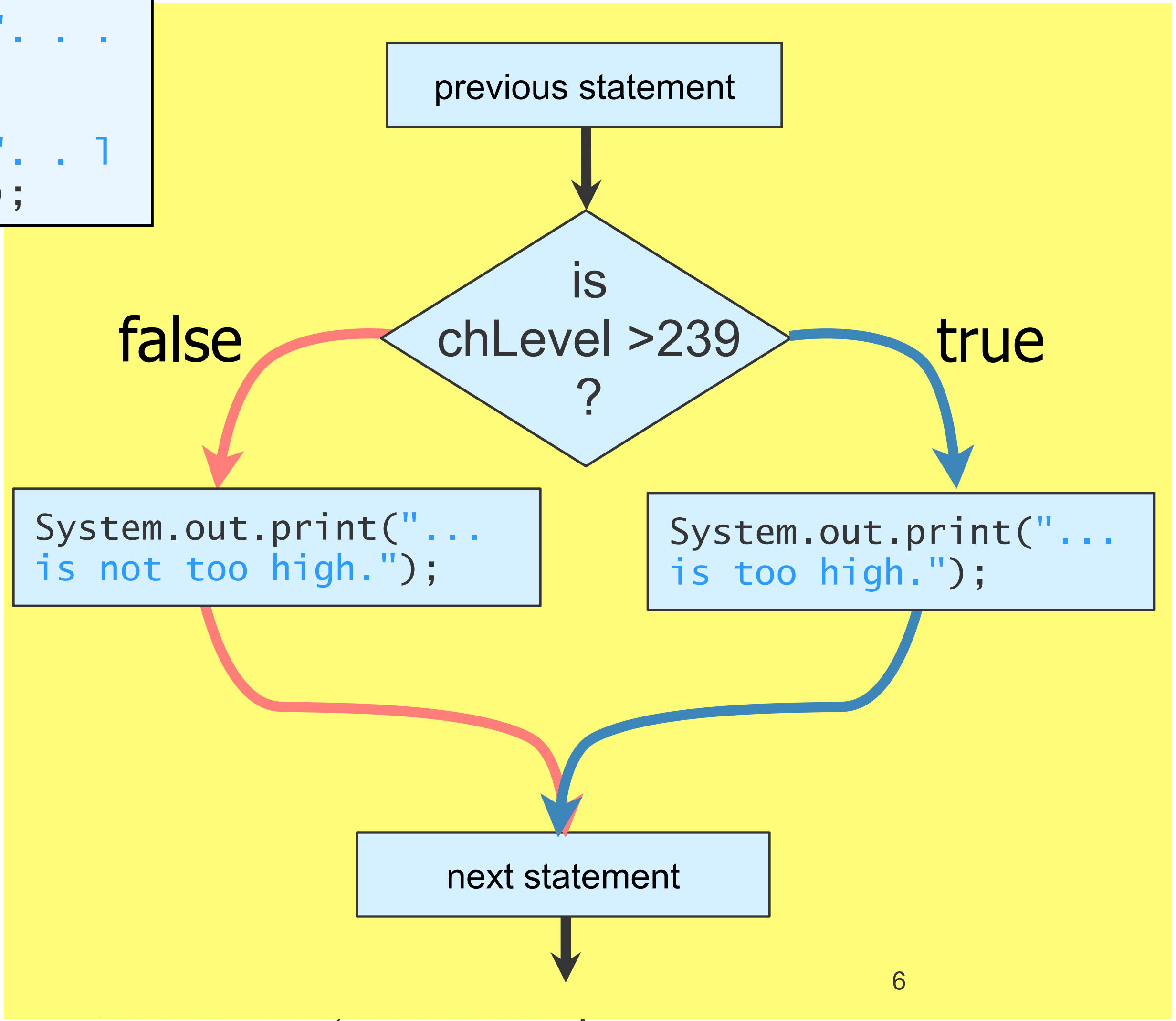
if-else Control Flow

```
if (chLevel > 239)
        System.out.print(". . .
        is too high.");
else
        System.out.print(". . l
        is not too high.");
```



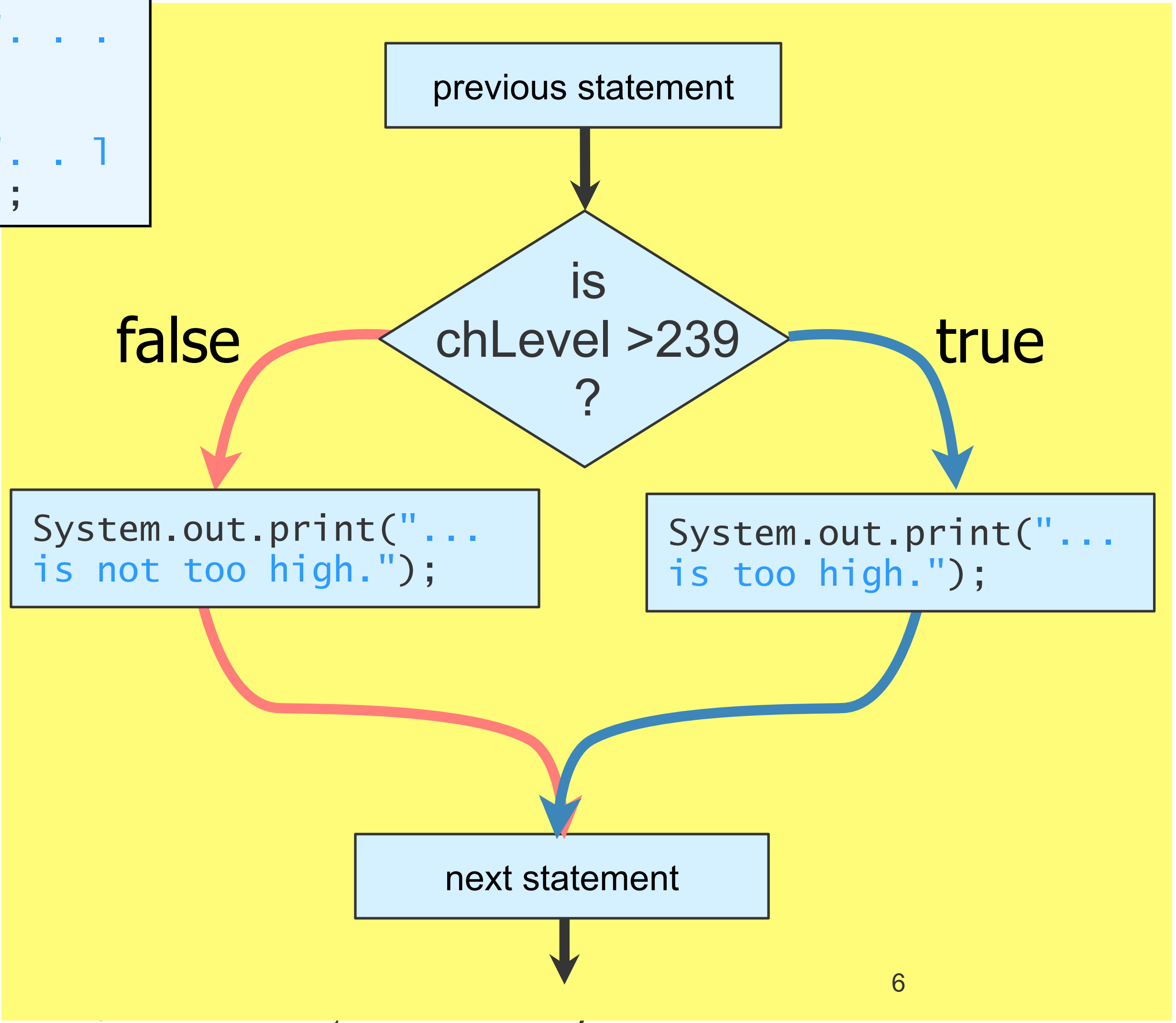
if-else Control Flow

```
if (chLevel > 239)
        System.out.print(". . .
        is too high.");
else
        System.out.print(". . l
        is not too high.");
```


previous statement
is chLevel >239 ?
false
true
System.out.print("... is not too high.");
System.out.print("... is too high.");
next statement

if-else Control Flow

```
if (chLevel > 239)
        System.out.print(". . .
        is too high.");
else
        System.out.print(". . l
        is not too high.");
```


previous statement
is chLevel >239 ?
false
true
System.out.print("... is not too high.");
System.out.print("... is too high.");
next statement

Depending upon the value of chLevel, one or the other branch is executed, not both.

if-else syntax

```
if ( <boolean expression> )
         if-statement;
else
         else-statement;
```

- The boolean expression is a special type of expression which can have one of two values: **true** or **false** values
- If the expression evaluates to **true**, the if-statement is executed; otherwise
- the else-statement is executed.

Multiple conditional statements

```
if ( <boolean expression> )
{
     if-statement1;
     if-statement2;
     ...
}
else
{
     else-statement1;
     else-statement2;
     else-statement3;
     ...
}
```

- We can have multiple statements for the if and/or else branches.
- Braces are used to combine multiple statements into a single block.

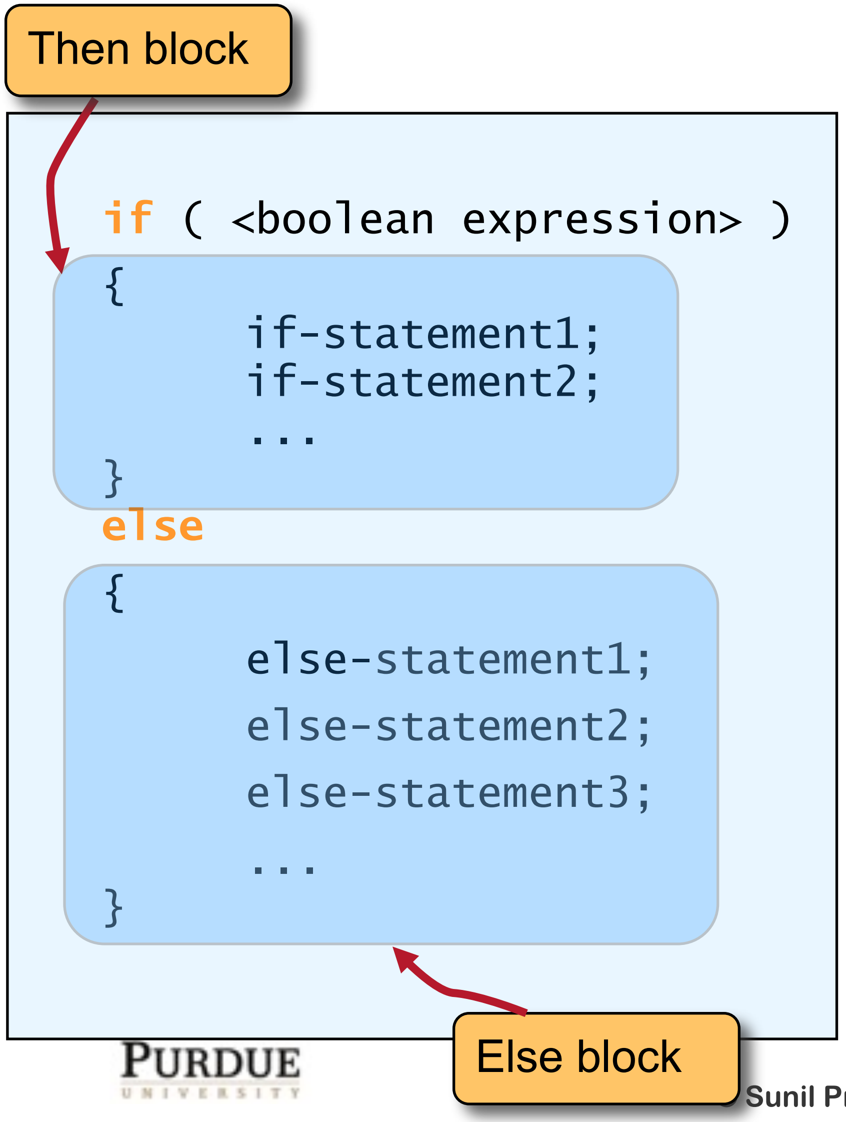
Multiple conditional statements

Then block

```
if ( <boolean expression> )
{
        if-statement1;
        if-statement2;
        ...
}
else
{
        else-statement1;
        else-statement2;
        else-statement3;
        ...
}
```

- We can have multiple statements for the if and/or else branches.
- Braces are used to combine multiple statements into a single block.

Multiple conditional statements

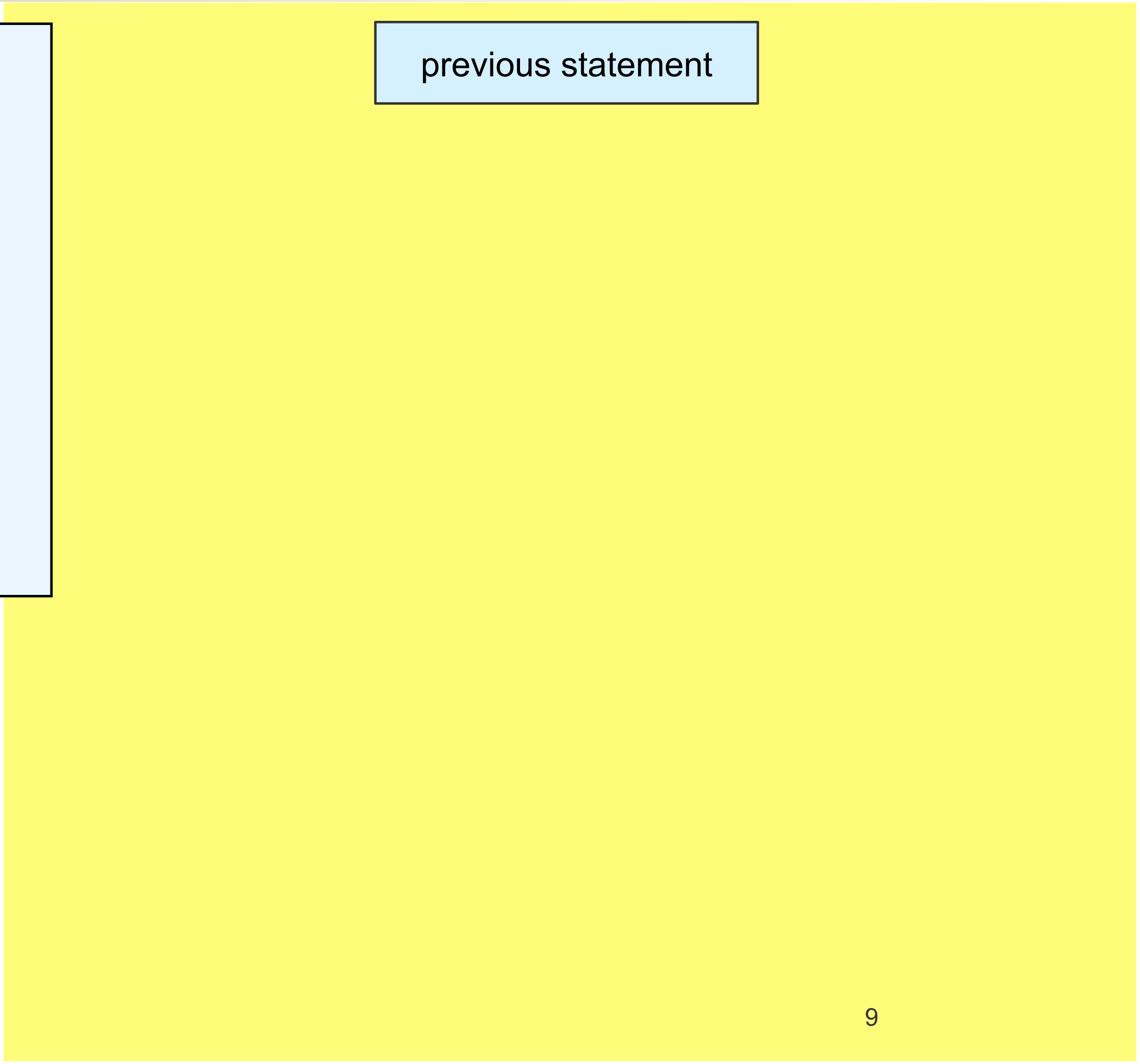


```
if ( <boolean expression> )
{
        if-statement1;
        if-statement2;
        ...
}
else
{
        else-statement1;
        else-statement2;
        else-statement3;
        ...
}
```

- We can have multiple statements for the if and/or else branches.
- Braces are used to combine multiple statements into a single block.

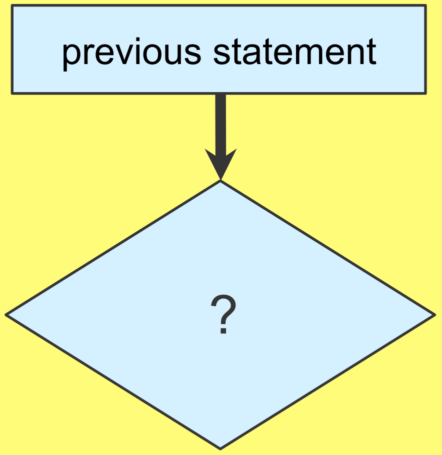
if-else Blocks Control Flow

```
if (<boolean expression>)
{
    if-statement1;
    if-statement2;
        …
}
else
{
    else-statement1;
    else-statement2;
    else-statement3;
        …
}
```



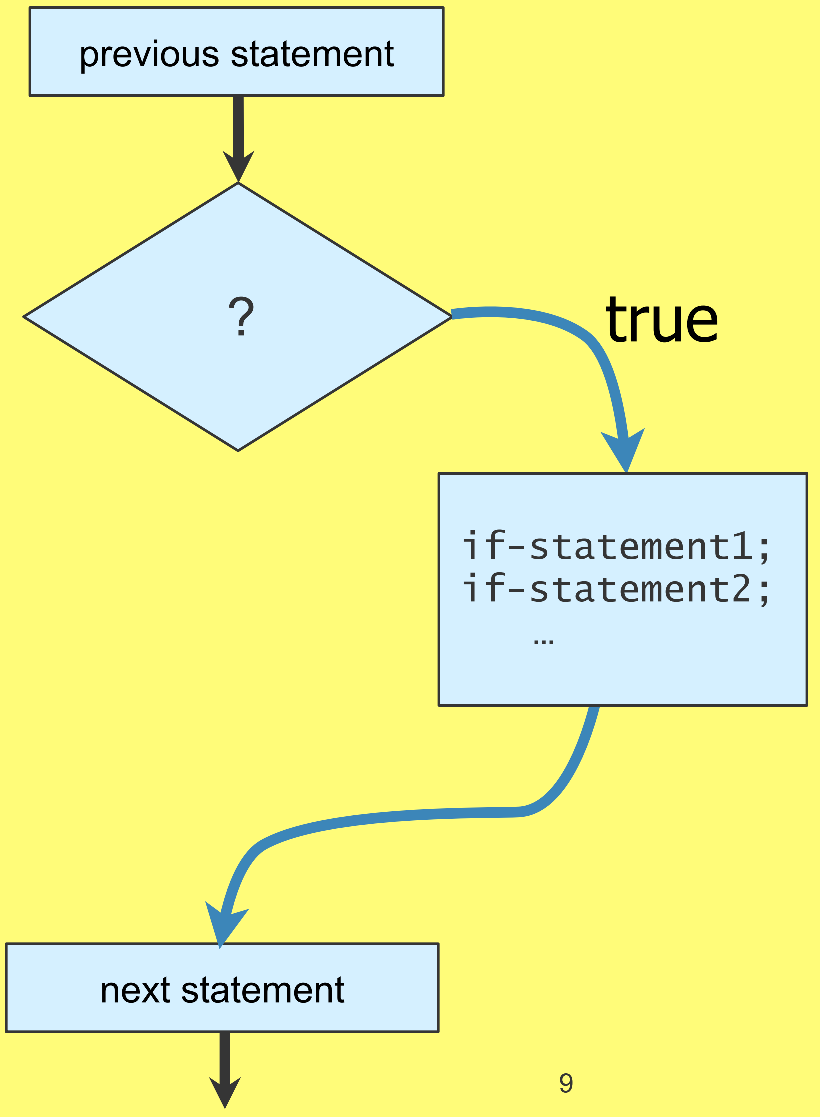
if-else Blocks Control Flow

```
if (<boolean expression>)
{
    if-statement1;
    if-statement2;
        …
}
else
{
    else-statement1;
    else-statement2;
    else-statement3;
        …
}
```



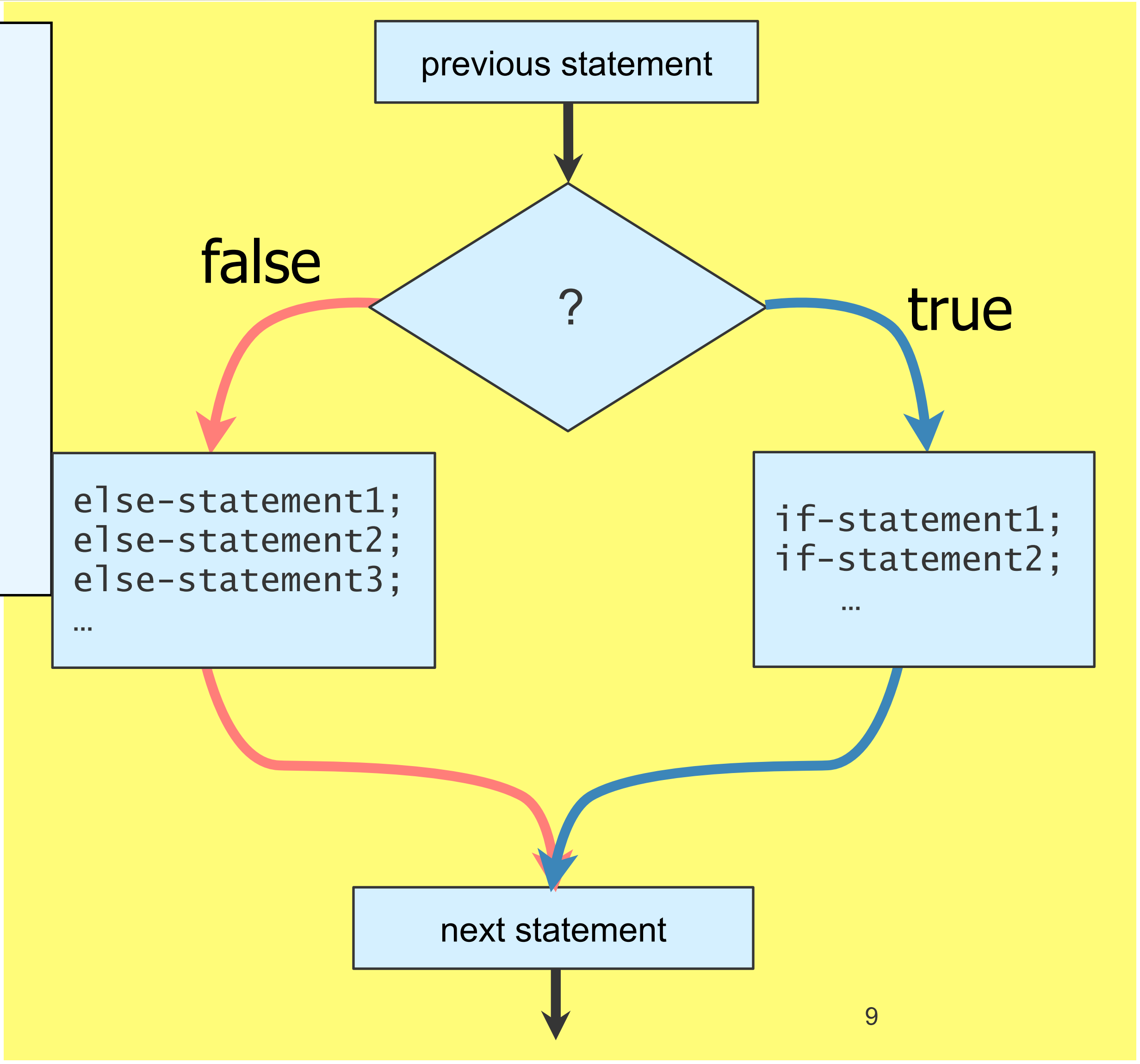
if-else Blocks Control Flow

```
if (<boolean expression>)
{
    if-statement1;
    if-statement2;
        …
}
else
{
    else-statement1;
    else-statement2;
    else-statement3;
        …
}
```



if-else Blocks Control Flow

```
if (<boolean expression>)
{
    if-statement1;
    if-statement2;
        …
}
else
{
    else-statement1;
    else-statement2;
    else-statement3;
        …
}
```



PURDUE UNIVERSITY

Solution

```
public class CholesterolCheck {
    public static void main(String[] args){
        int chLevel;
        chLevel = Integer.parseInt(JOptionPane.showInputDialog(
                        null, "Enter your cholesterol measure"));

        if(chLevel > 239) {

            System.out.println("Your cholesterol level is too high.");
            System.out.println ("You should probably see a doctor.");

        } else {

            System.out.println("Your cholesterol level is not too high.");
            System.out.println("Don't forget to exercise.");

        }

    }
}
```

Boolean Expressions

- boolean is a primitive data type.
- A boolean value can only be either **true** or **false**
- A simple boolean expression compares two values using a relational operator, e.g.,
  - chLevel > 239
  - height < weight
  - gpa == 3.0
- The operands can be either variables or literal values.

Relational Operators

- The following operators can be used to compare numeric data types:

Relational Operator	Meaning
>	Greater than
<	Less than
==	Equal to
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to

Do not confuse with assignment (=).

Complex boolean expressions

- Boolean expressions can be combined using boolean operators to form more complex expressions.
  - Analogous to normal conditional statements.
- For example,
  - given three **int** variables i,j, and k:

```
(i > j) && (k == 5)
```

  - evaluates to **true** only if the value stored in i is greater than the value stored in j AND the value stored in k is equal to 5; **false** otherwise.

Boolean Operators

- Boolean operators take boolean expressions as operands.

Boolean Operator	Meaning
&&	AND
||	OR
!	Not (negation). Takes only one operand
^	Exclusive-OR

Boolean Operators

- Boolean operators take boolean expressions as operands.

Boolean Operator	Meaning
&&	AND
||	OR
!	Not (negation). Takes only one operand
^	Exclusive-OR

These are two "pipe" characters

Boolean Operators (contd)

Boolean Operators (contd)

- bool1 && bool2
  - is **true** if both bool1 and bool2 are **true**;
  - otherwise it is **false**
    - (x > 2) && (x<10) is **true** for x=3; **false** for x=11;

Boolean Operators (contd)

- bool1 && bool2
  - is **true** if both bool1 and bool2 are **true**;
  - otherwise it is **false**
    - (x > 2) && (x<10) is **true** for x=3; **false** for x=11;

- bool1 || bool2
  - is **true** if either bool1 or bool2 (or both) are **true**;
  - otherwise it is **false**
    - (x>2) || (x<10) is always true.

Boolean Operators (contd)

Boolean Operators (contd)

- !bool1
  - is **true** if bool1 is **false**,
  - and **false** if bool1 is **true**
    - !(x>2) is **true** for x=1; and **false** for x=3;

Boolean Operators (contd)

- !bool1
  - is **true** if bool1 is **false**,
  - and **false** if bool1 is **true**
    - !(x>2) is **true** for x=1; and **false** for x=3;

- bool1 ^ bool2
  - is **true** if bool1 and bool2 are **different**;
  - otherwise it is **false**
    - (x>2) ^ (x<10) is false for x=3; and true for x = 11;

Definition of Boolean Operators

- Truth table for boolean operators

p	q	p && q	p || q	!p	p^q
false	false	false	false	true	false
false	true	false	true	true	true
true	false	false	true	false	true
true	true	true	true	false	false

- Sometimes true and false are represented by 1 and 0 (NOT in Java).
- In C and C++, 0 is **false**, everything else is **true**.

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

```
i < j
```

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

```
i < j
```

```
f >= i
```

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

```
i < j
```

```
f >= i
```

```
d > 9.3
```

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

```
i < j
```

```
f >= i
```

```
d > 9.3
```

```
2 == c
```

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

`i < j`

`f >= i`

`d > 9.3`

`2 == c`

`j != i`

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

```
i < j
```

```
f >= i
```

```
d > 9.3
```

```
2 == c
```

```
j != i
```

```
g <= (b*c + d)
```

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

```
i < j
```

```
f >= i
```

```
d > 9.3
```

```
2 == c
```

```
j != i
```

```
g <= (b*c + d)
```

```
(i > j)  &&  (f >= i)
```

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

```
i < j
```

```
f >= i
```

```
d > 9.3
```

```
2 == c
```

```
j != i
```

```
g <= (b*c + d)
```

```
(i > j)  &&  (f >= i)
```

```
(d > 9.3) || (2 != d)
```

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

```
i < j
```

```
f >= i
```

```
d > 9.3
```

```
2 == c
```

```
j != i
```

```
g <= (b*c + d)
```

```
(i > j)  &&  (f >= i)
```

```
(d > 9.3) || (2 != d)
```

```
!(c <= j) ^ (j != i)
```

Examples of boolean expressions.

```
int i, j;
byte b, c;
float f, g;
double d, e;
```

```
i < j
```

```
f >= i
```

```
d > 9.3
```

```
2 == c
```

```
j != i
```

```
g <= (b*c + d)
```

```
(i > j)  &&  (f >= i)
```

```
(d > 9.3) || (2 != d)
```

```
!(c <= j) ^ (j != i)
```

```
((i > j)  &&  (f >= i)) || ((d > 9.3) || (2 != d)) ^ (!(c <= j) ^ (j != i))
```

Problem

- *Write a program that tells a patient how to interpret their total cholesterol measure. The measure is an integer. A cholesterol measure*
    - *Less than 200 is "Desirable"*
    - *200-239 is "Mildly High"*
    - *240 and above is "High"*
- *Your program should read in the measure and output an appropriate evaluation.*

The Nested-if Statement

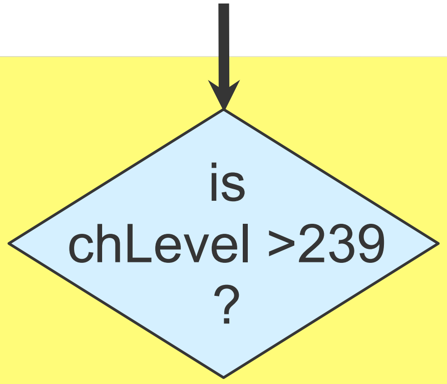
- The `then` and `else` block of an if statement can contain any valid statements, including other if statements. An if statement containing another if statement is called a nested-if statement.

```
if (chLevel > 239)
   System.out.print(". . . is too high.");
else
   if (chLevel > 199)
      System.out.print(". . is mildly high.");
   else
      System.out.print(". . is normal.");
```

Sample control flow

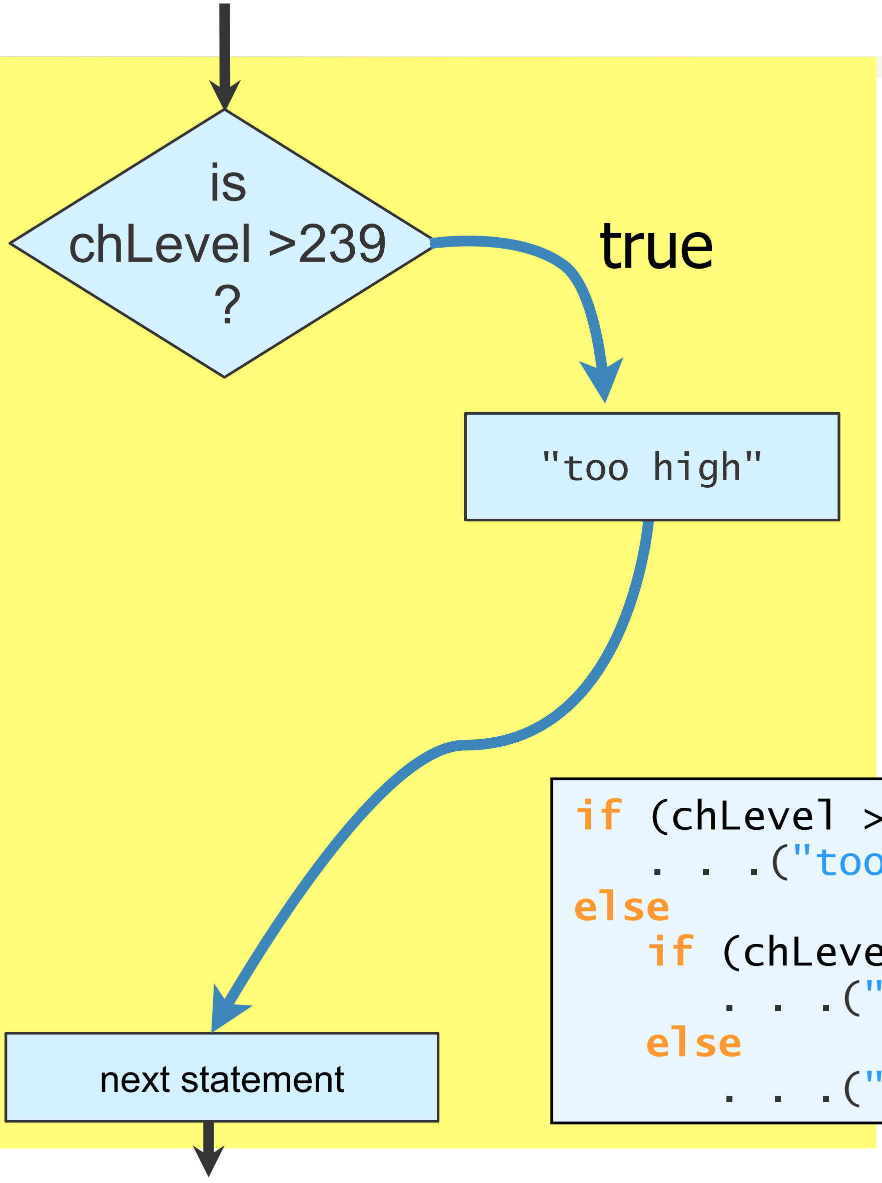
```
if (chLevel > 239)
   . . .("too high.");
else
   if (chLevel > 199)
      . . .("mildly high.");
   else
      . . .("normal.");
```

Sample control flow



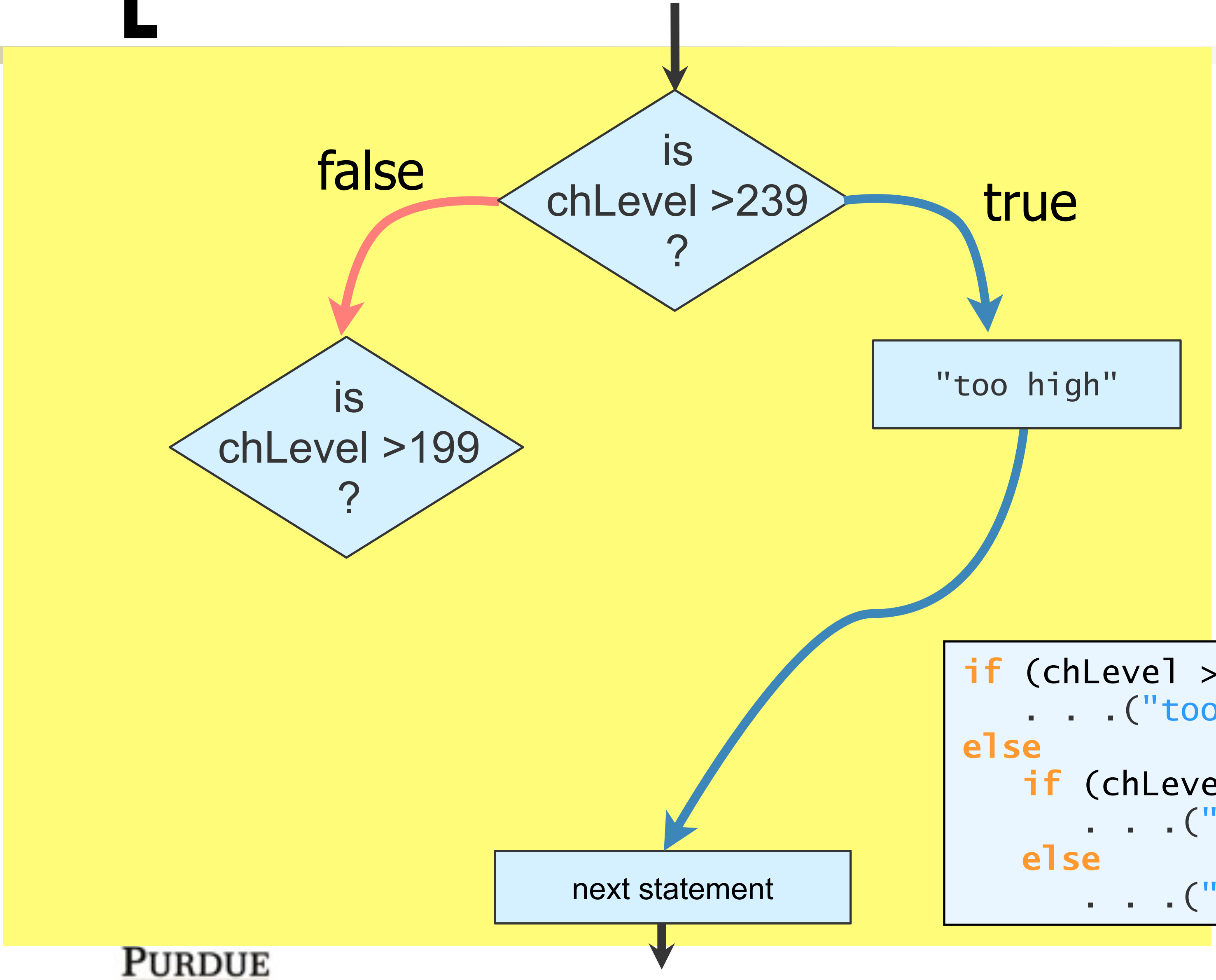
```
if (chLevel > 239)
    . . .("too high.");
else
    if (chLevel > 199)
        . . .("mildly high.");
    else
        . . .("normal.");
```

Sample control flow



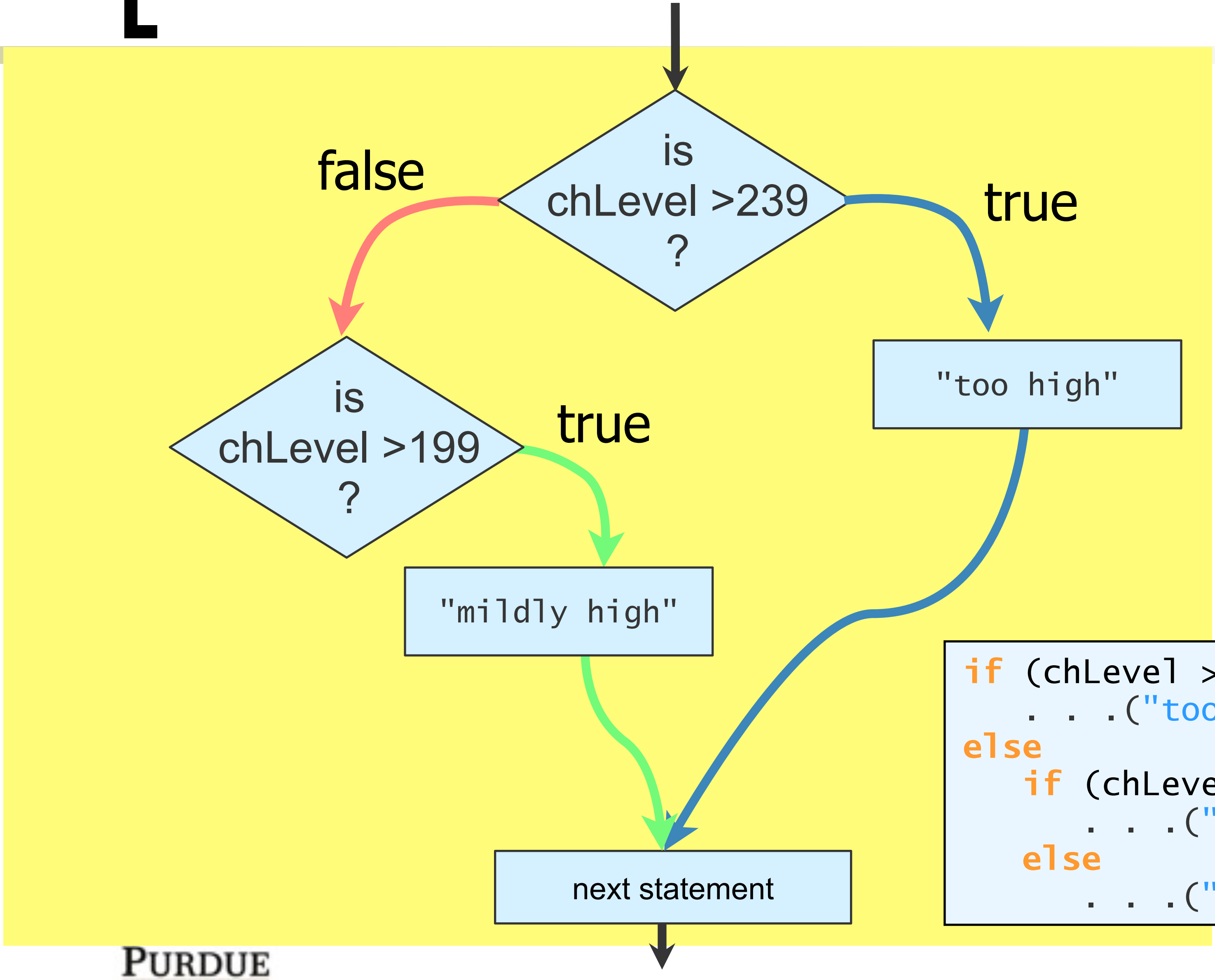
```
if (chLevel > 239)
    . . .("too high.");
else
    if (chLevel > 199)
        . . .("mildly high.");
    else
        . . .("normal.");
```

Sample control flow



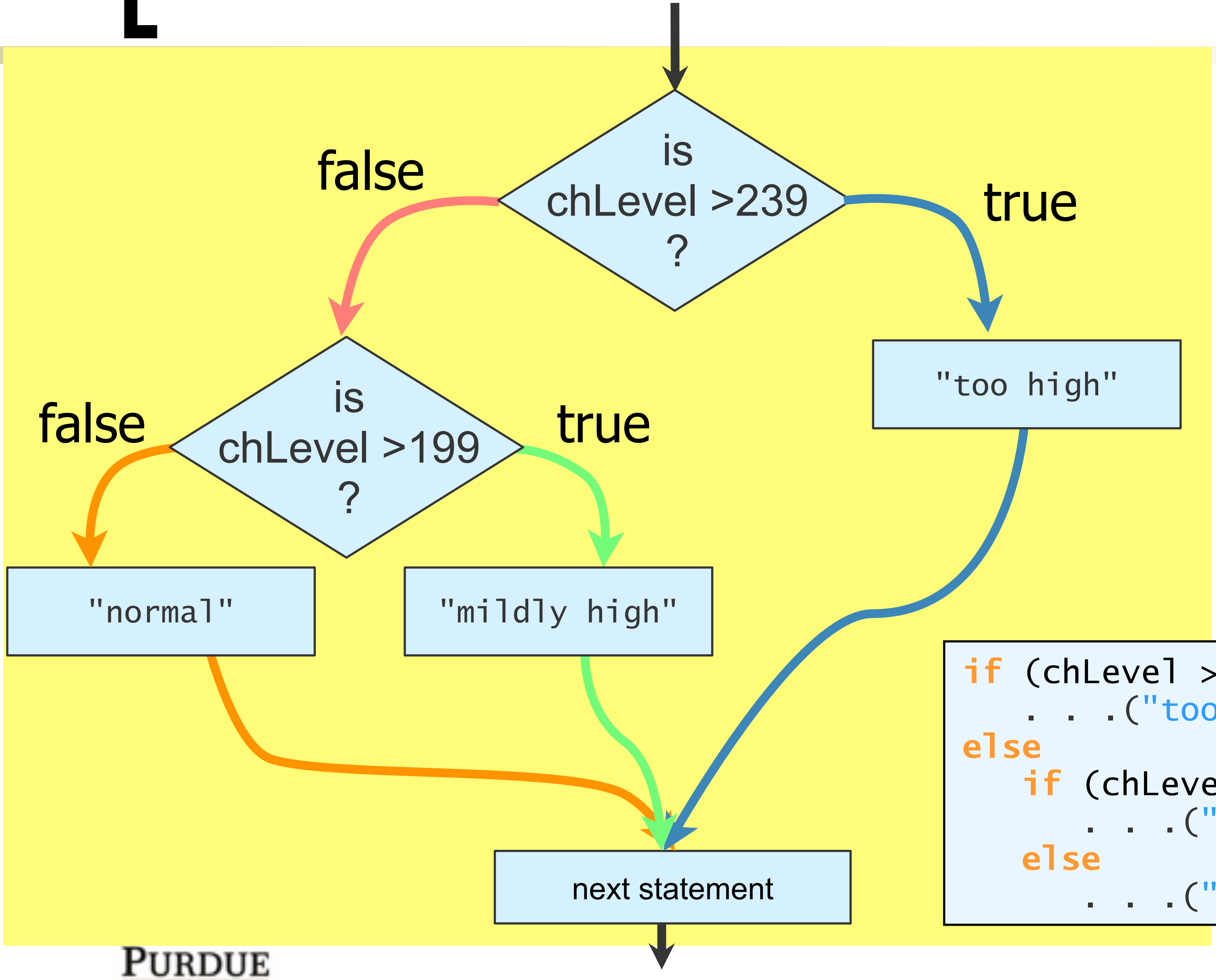
```
if (chLevel > 239)
    . . .("too high.");
else
    if (chLevel > 199)
        . . .("mildly high.");
    else
        . . .("normal.");
```

Sample control flow



```
if (chLevel > 239)
    . . .("too high.");
else
    if (chLevel > 199)
        . . .("mildly high.");
    else
        . . .("normal.");
```

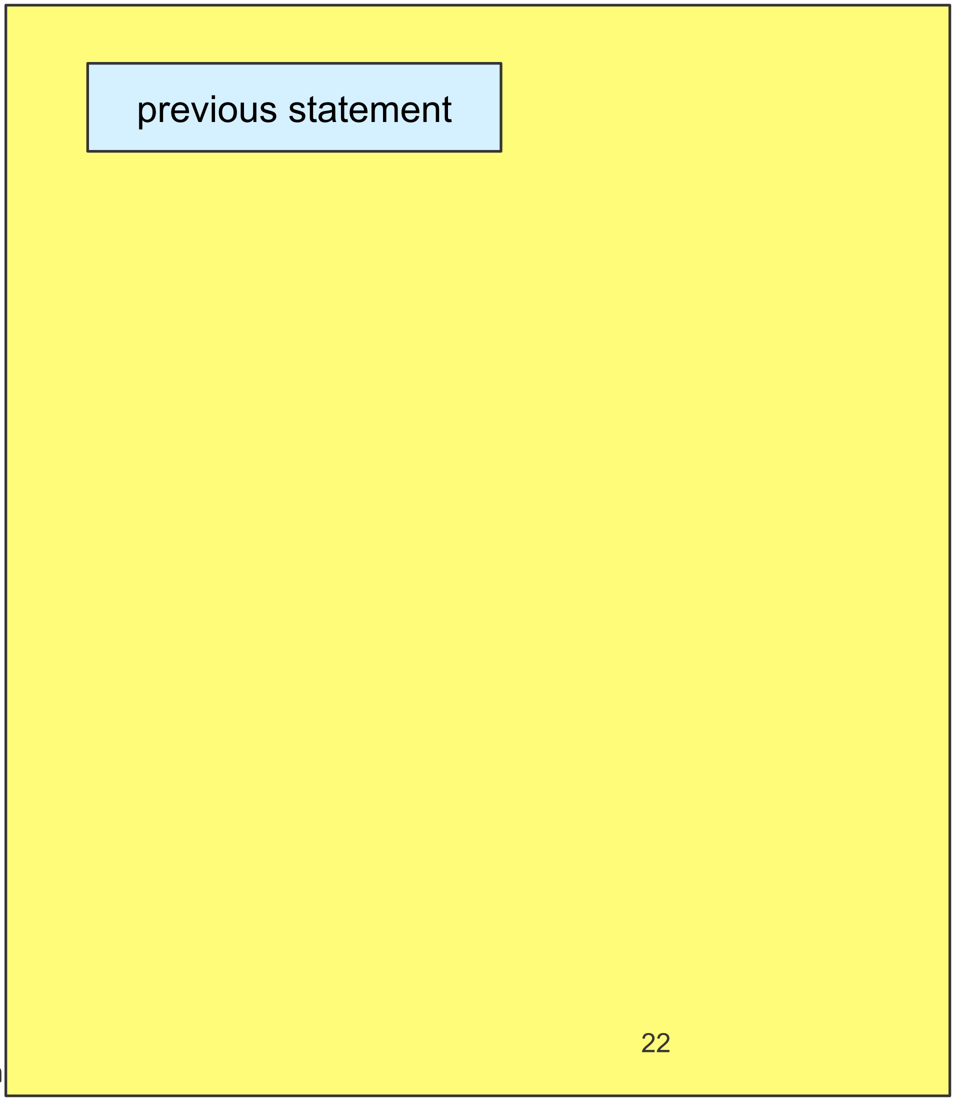
Sample control flow



```
if (chLevel > 239)
    . . .("too high.");
else
    if (chLevel > 199)
        . . .("mildly high.");
    else
        . . .("normal.");
```

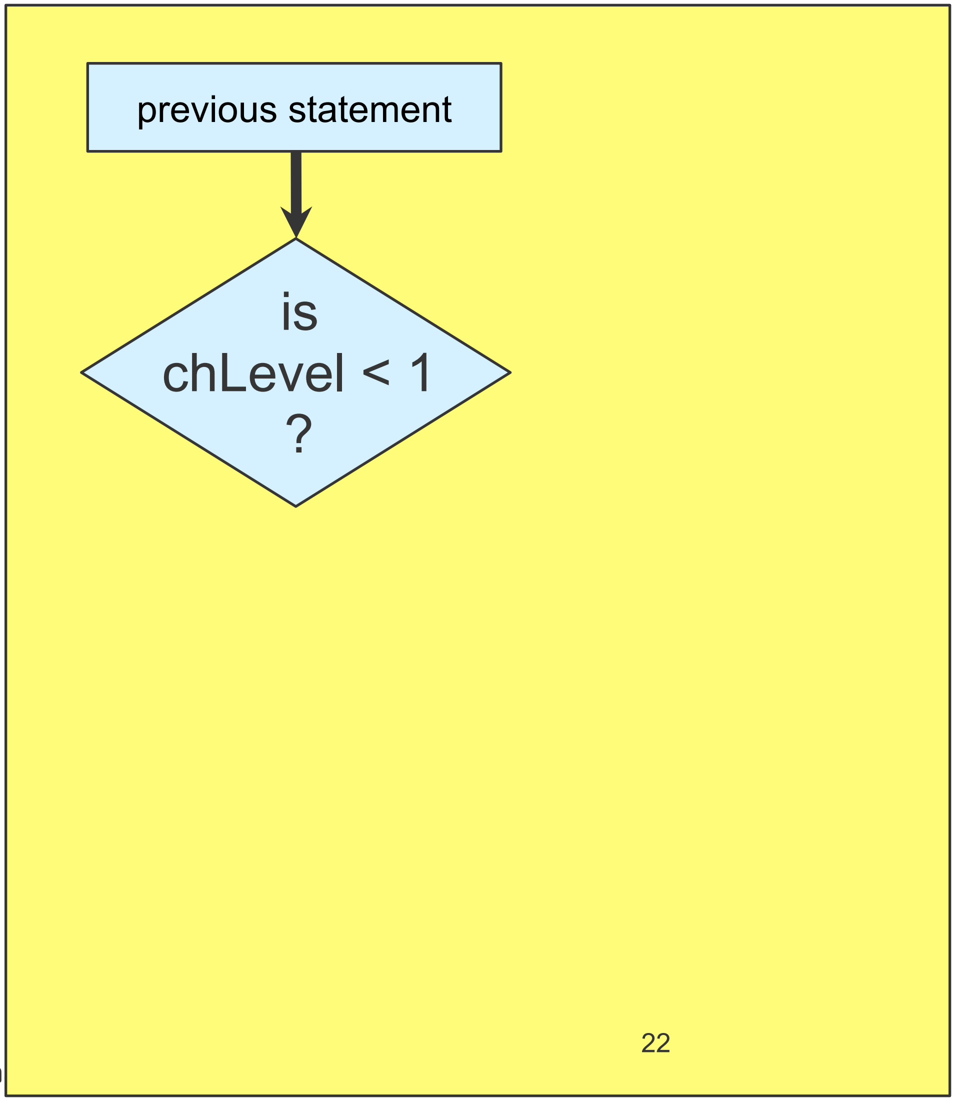
else is Not Required

```
if (chLevel < 1){
    System.out.print("There is an
    error in your input");
}
...
```



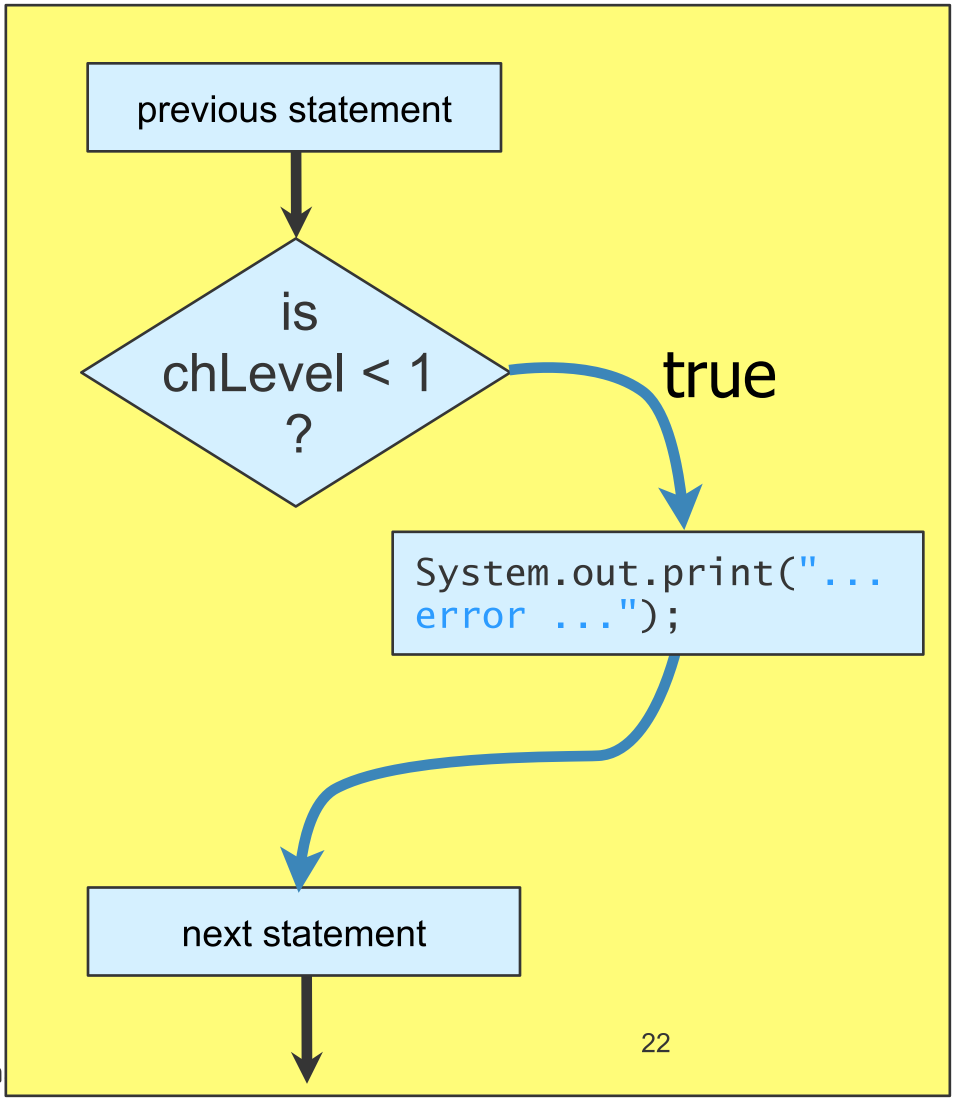
else is Not Required

```
if (chLevel < 1){
    System.out.print("There is an
    error in your input");
}
...
```



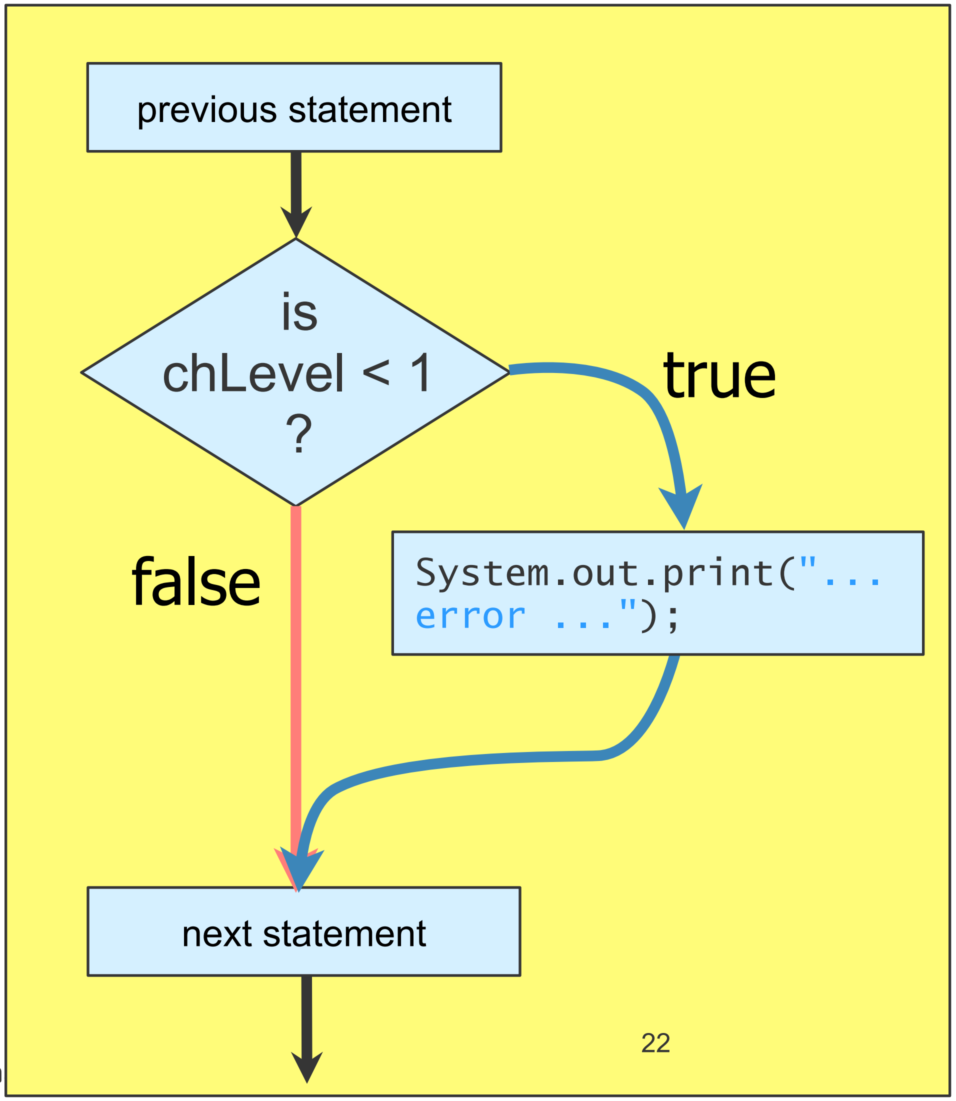
else is Not Required

```
if (chLevel < 1){
    System.out.print("There is an
    error in your input");
}
...
```



else is Not Required

```
if (chLevel < 1){
    System.out.print("There is an
    error in your input");
}
...
```



Caution: Dangling else

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Mildly High");
```

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
else
   System.out.print("Normal");
```

PURDUE UNIVERSITY

Caution: Dangling else

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Mildly High");
```

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
else
   System.out.print("Normal");
```

Each else paired with nearest unmatched if -- use braces to change this as needed.

Caution: Dangling else

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Mildly High");
```

Same as

```
if (chLevel > 199){
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Mildly High");
}
```

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
else
   System.out.print("Normal");
```

Each else paired with nearest unmatched if -- use braces to change this as needed.

PURDUE UNIVERSITY

Caution: Dangling else

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Mildly High");
```

Same as

```
if (chLevel > 199){
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Mildly High");
}
```

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
else
   System.out.print("Normal");
```

Each else paired with nearest unmatched if -- use braces to change this as needed.

Caution: Dangling else

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Mildly High");
```

Same as

```
if (chLevel > 199){
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Mildly High");
}
```

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
else
   System.out.print("Normal");
```

Same as

```
if (chLevel > 199) {
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Normal");
}
```

Each else paired with nearest unmatched if -- use braces to change this as needed.

Caution: Dangling else

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Mildly High");
```

Same as

```
if (chLevel > 199){
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Mildly High");
}
```

```
if (chLevel > 199)
   if (chLevel > 239)
      System.out.print("Too High");
else
   System.out.print("Normal");
```

Same as

```
if (chLevel > 199) {
   if (chLevel > 239)
      System.out.print("Too High");
   else
      System.out.print("Normal");
}
```

```
if (chLevel > 199) {
   if (chLevel > 239)
      System.out.print("Too High");
} else
      System.out.print("Normal");
```

Each else paired with nearest unmatched if -- use braces to change this as needed.

Boolean Variables

- Boolean values can be stored in **boolean** variables -- a primitive datatype.
- Can be used in boolean expressions.

Boolean Variables

- Boolean values can be stored in **boolean** variables -- a primitive datatype.
- Can be used in boolean expressions.

```
boolean hasWon,isFinalLevel;

isFinalLevel = false;
…
isFinalLevel = (gameLevel == 10);
hasWon = (numberOfZombies == 0);

if (hasWon)
   if (isFinalLevel)
      System.out.println("WOW -- you beat the game!");
   else
      startNextLevel();
else
   restartSameLevel();
```

Boolean Methods

- A method that returns a boolean value is a Boolean method.
- A call to this method can be used as a boolean value.

Boolean Methods

- A method that returns a boolean value is a Boolean method.
- A call to this method can be used as a boolean value.

```
public boolean isGameOver(){
  if((numberOfHumans < 1) || (numberOfZombies<1))
      return true;
   else
      return false;
}
```

Boolean Methods

- A method that returns a boolean value is a Boolean method.
- A call to this method can be used as a boolean value.

```
public boolean isGameOver(){
  if((numberOfHumans < 1) || (numberOfZombies<1))
      return true;
   else
      return false;
}
```

```
if( isGameOver() )
   if(numberOfZombies < 1)
      System.out.println("You WON!!");
   else
      System.out.println("Sorry, you lost!!");
else
   System.out.println("Battle on…");
```

PURDUE
UNIVERSITY

Operator Precedence Rules

Group	Operator	Order
Subexpresion	( )	**Innermost first**
Unary operators	- , !	**Right to Left**
Unary operators	*, /, %	Left to Right
Additive	+ , -	Left to Right
Relational	<, <=, >, <,	Left to Right
Equality	!=, ==	Left to Right
Boolean AND	&&	Left to Right
Boolean OR	||	Left to Right
Assignment	=	**Right to Left**

PURDUE UNIVERSITY

Announcements

- Midterm exam 1:
  - September 26, 8:00 – 9:00pm
  - Two rooms EE129 & FRNY G140
  - You will be assigned a room
  - Coverage: upto Week 5.
  - Closed book/notes. Can bring one sheet.
  - Sample exams are on website.

Problem

- *Write a program that classifies triangles*
  - *by their sides*
  - *by their angles*
- *Write a program that classifies quadrilaterals by their sides and one angle*
  - *consider only parallelograms, rectangles, squares and rhombi.*

TriangleClassifier

```
class TriangleClassifier {
  public static void main(String args[]) {
   int side1, side2, side3;
   String type;
     . . . // read in all three side lengths
   if (side1 == side2)
     if (side1 == side3)
       type = "Equilateral";
     else
       type = "Isosceles";
   else
     type = "Scalene";
   System.out.println("This is a " + type + "
   triangle.");
  }
}
```

PURDUE UNIVERSITY

TriangleClassifier

```
class TriangleClassifier {
  public static void main(String args[]) {
   int side1, side2, side3;
   String type;
     . . . // read in all three side lengths

   if (side1 == side2)
      if (side1 == side3)
         type = "Equilateral";
      else
         type = "Isosceles";
   else
      type = "Scalene";
   System.out.println("This is a " + type + "
   triangle.");
  }
}
```

Not quite!!

TriangleClassifier

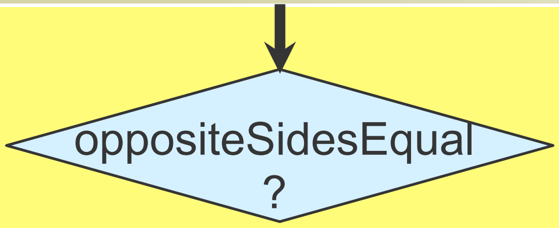
```
class TriangleClassifier {
  public static void main(String args[]) {
    int side1, side2, side3;
    String type;
     . . . // read in all three side lengths
    if (side1 == side2)
       if (side1 == side3)
          type = "Equilateral";
       else
          type = "Isosceles";

     else
        if ((side2 == side3)||(side1 == side3))
          type = "Isosceles";
       else
           type = "Scalene";
     System.out.println("This is a " + type + " triangle.");
  }
}
```

TriangleClassifier2

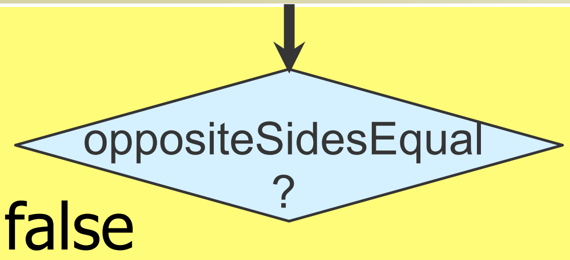
```
class TriangleClassifier2 {
  public static void main(String args[]) {
     double angle1, angle2, angle3, maxAngle;
     String type;
      . . . // read in all three angles
     maxAngle = Math.max(angle1, Math.max(angle2, angle3));
     if (Math.abs(maxAngle - 90.0) < 0.0000001)
        type = " right-angled";
     else
        if (maxAngle > 90)
           type = "n obtuse";

    else
        type = "n acute";
     System.out.println("This is a" + type + " triangle.");
  }
}
```

Quadrilateral Logic

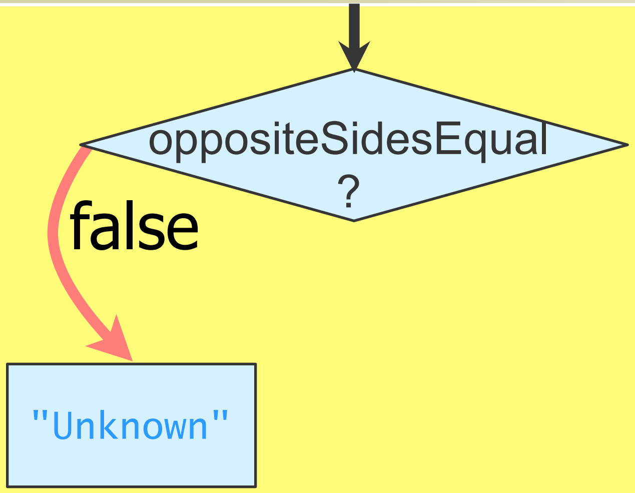
Quadrilateral Logic



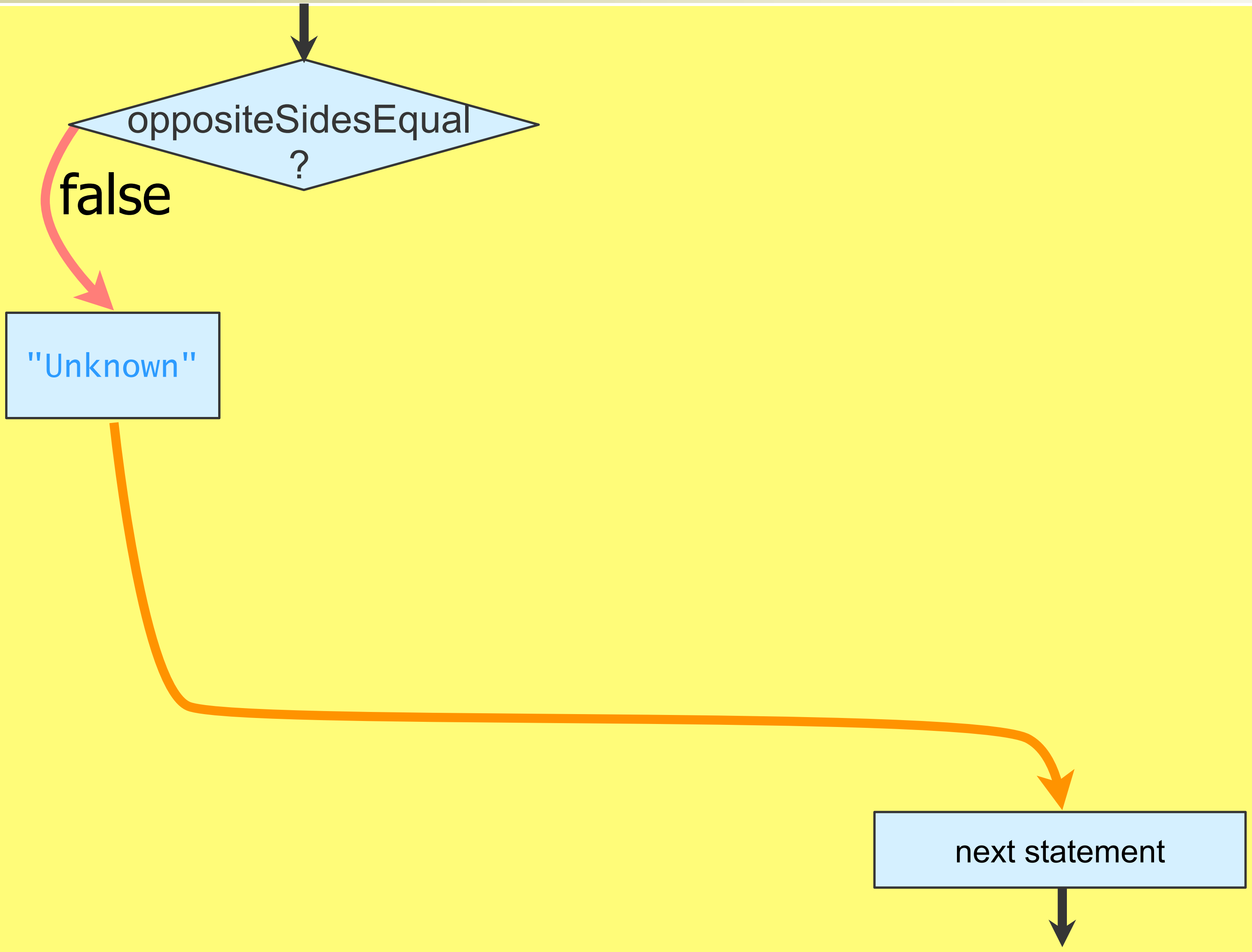
Quadrilateral Logic



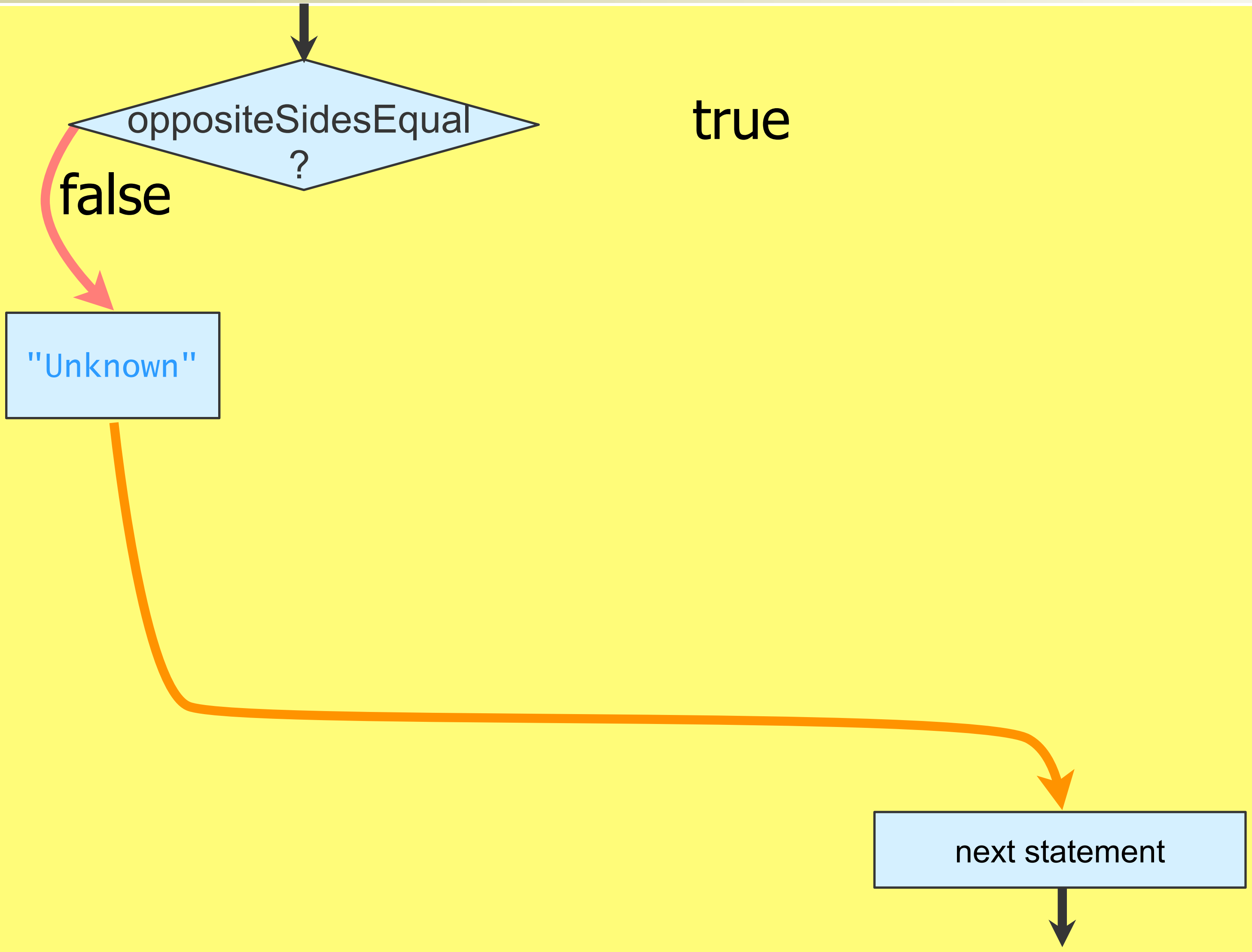
Quadrilateral Logic



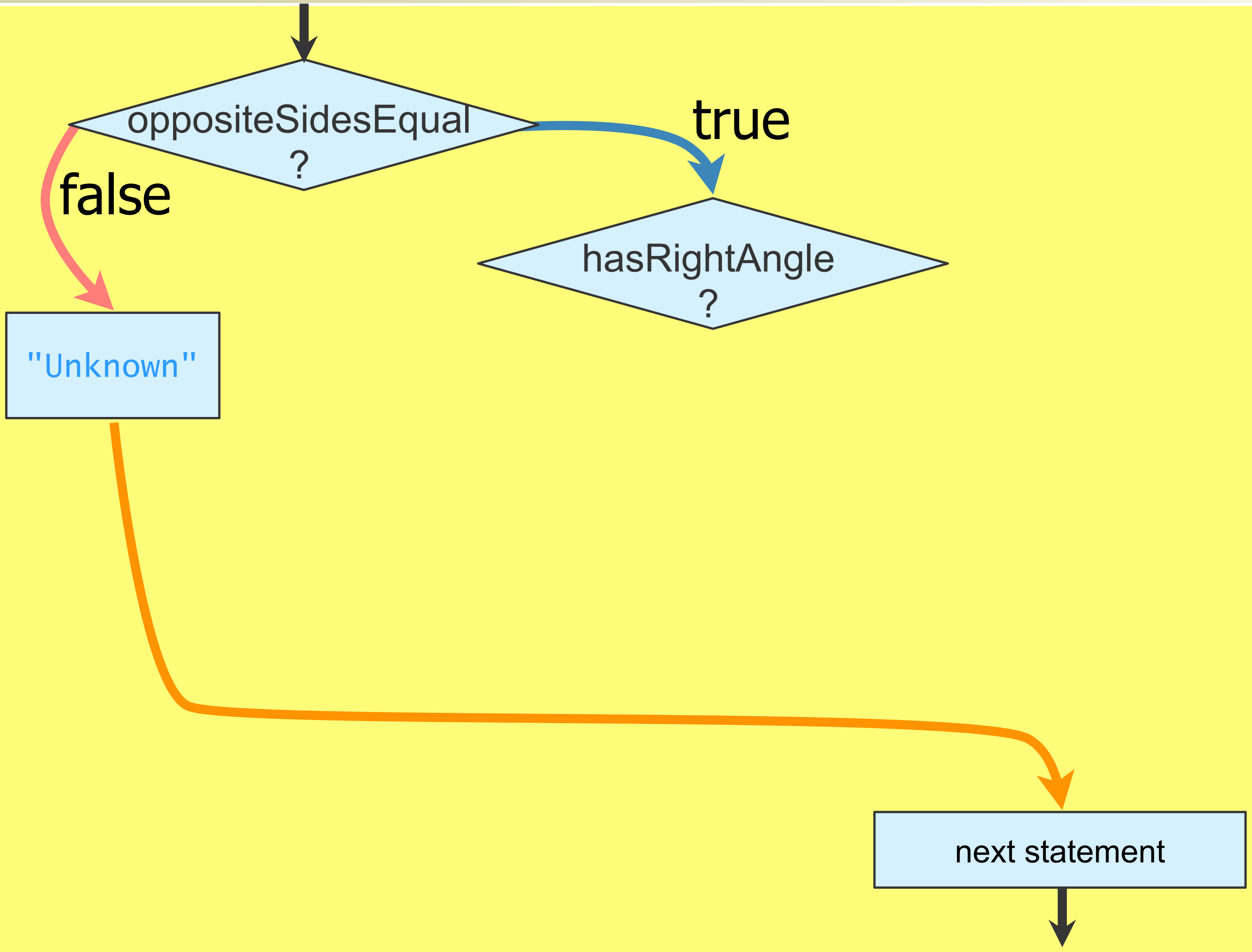
Quadrilateral Logic



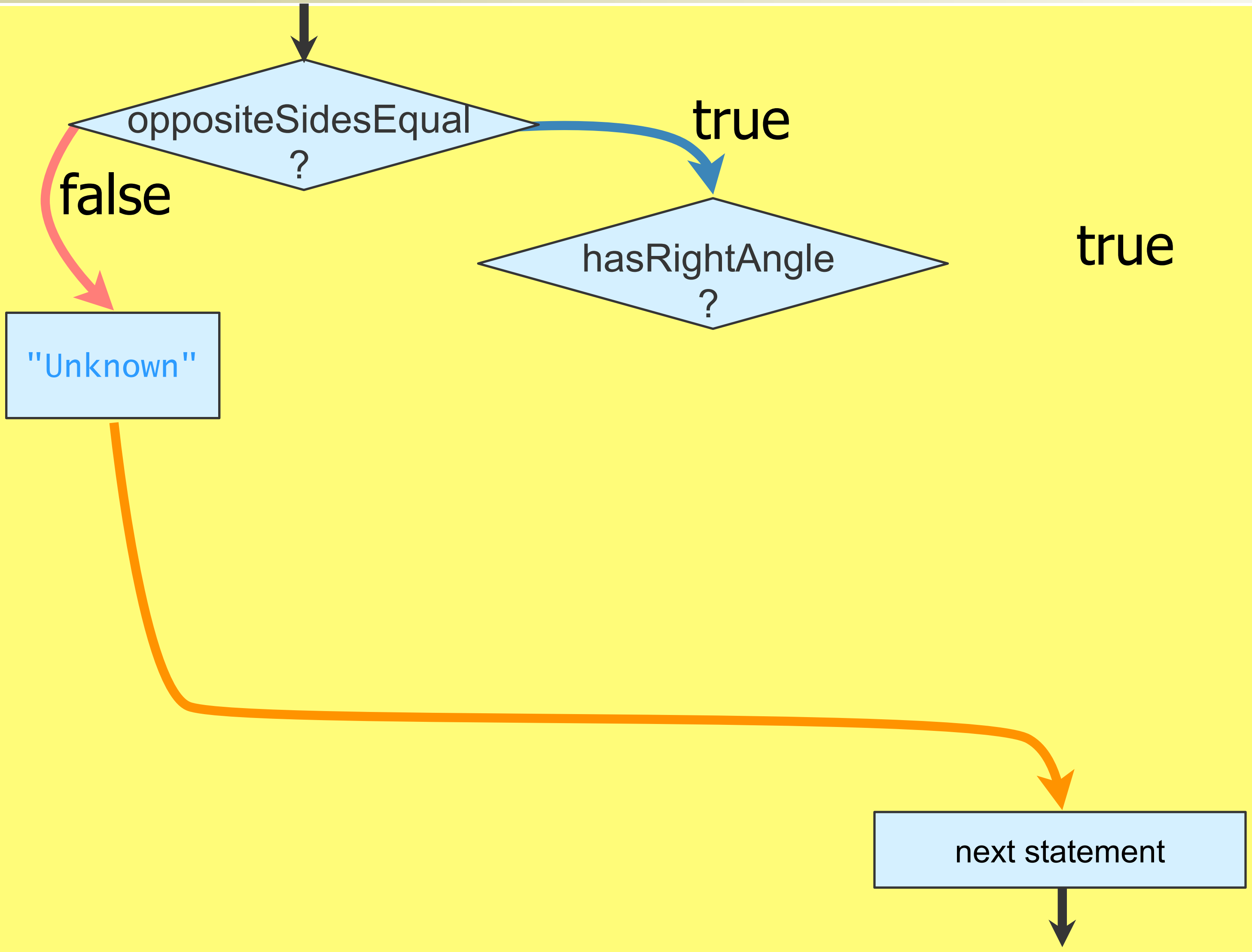
Quadrilateral Logic



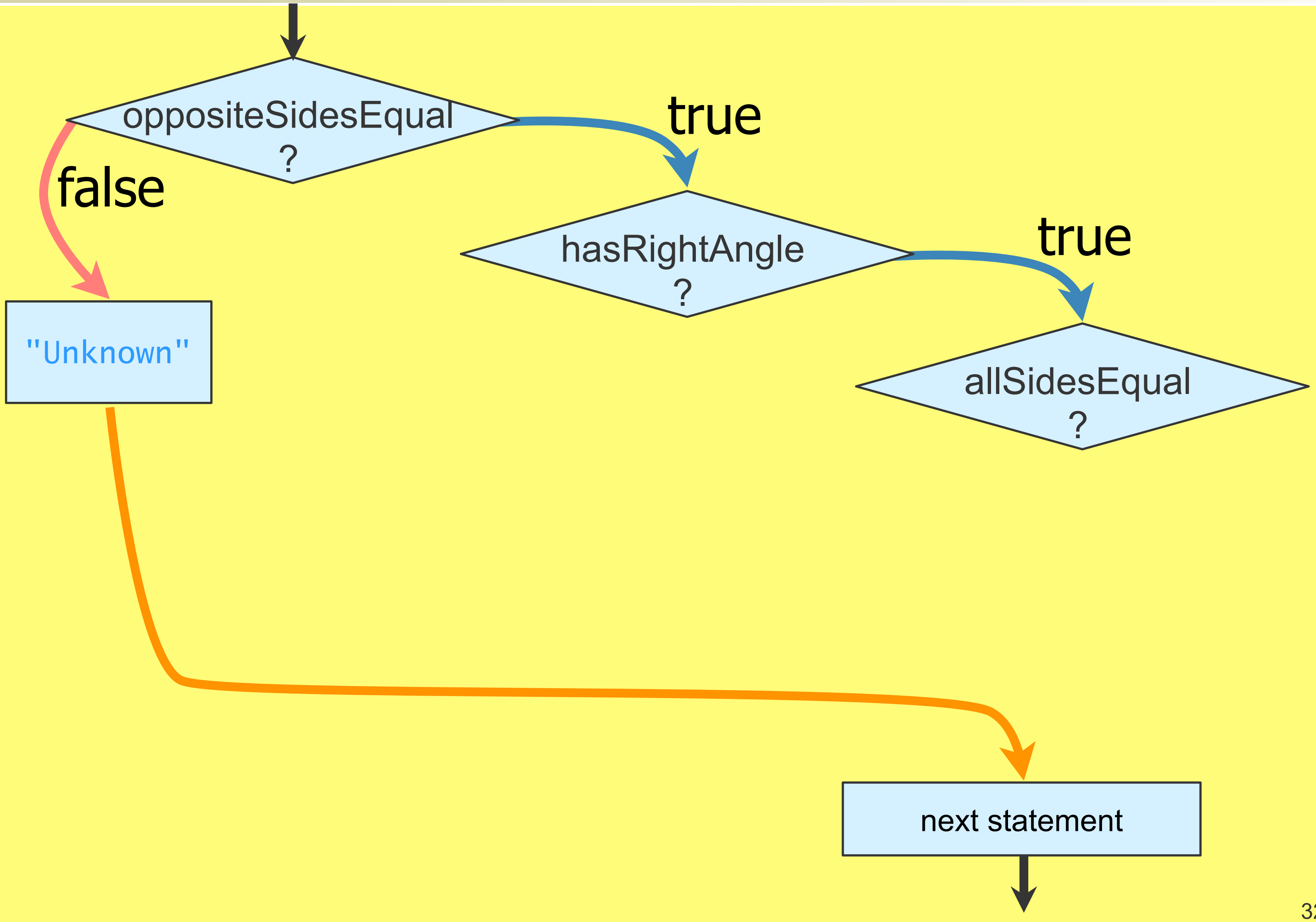
Quadrilateral Logic



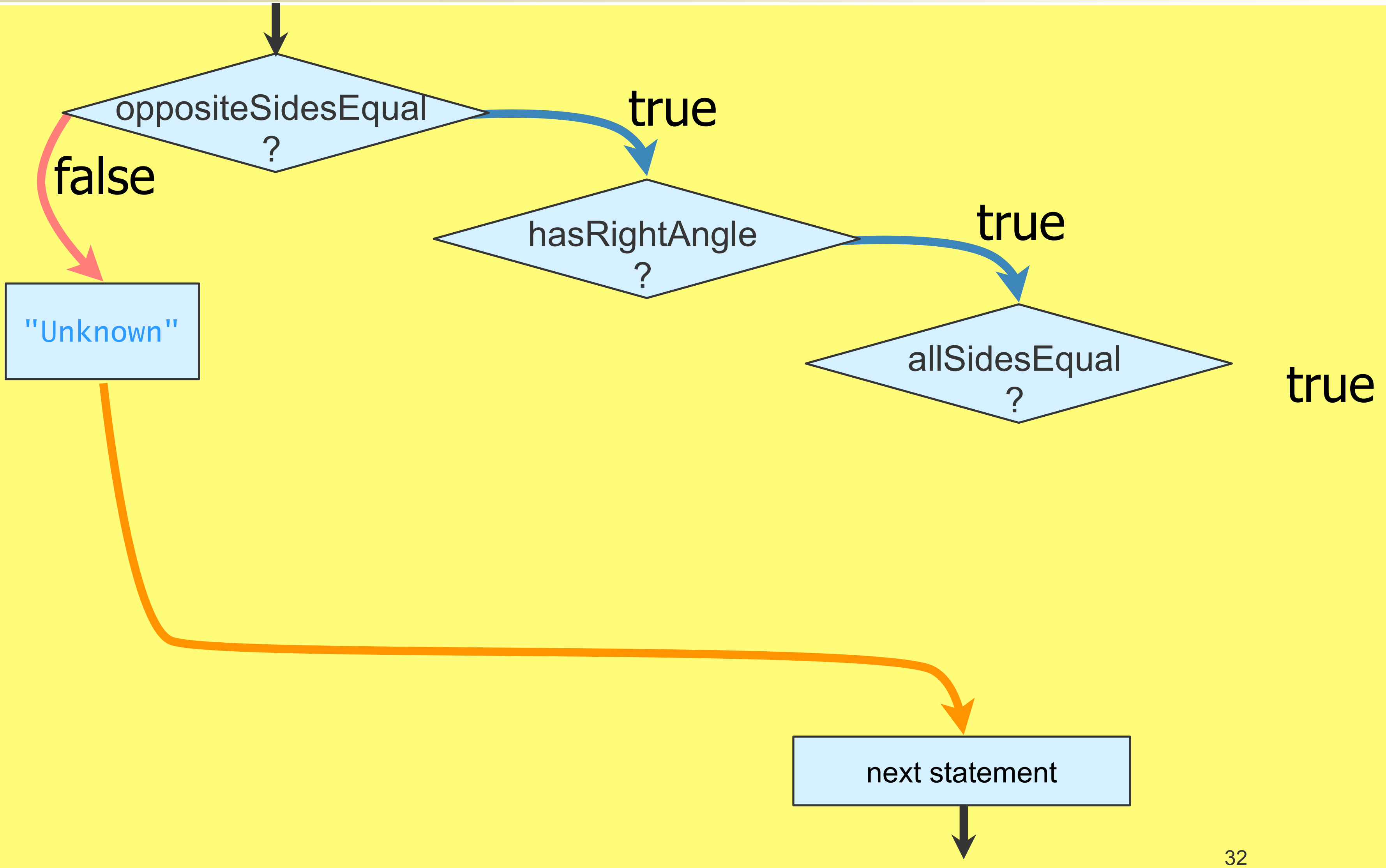
Quadrilateral Logic



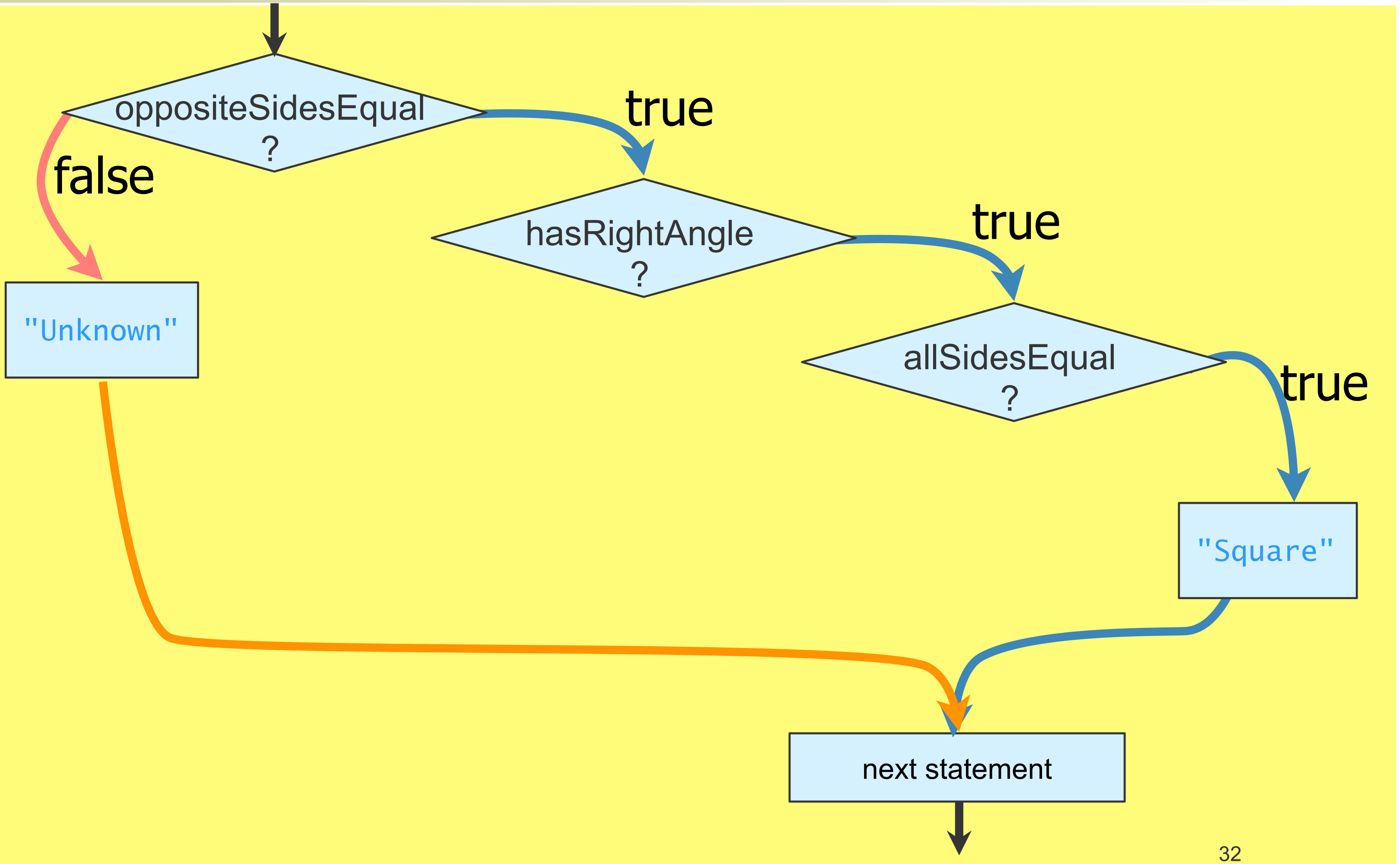
Quadrilateral Logic



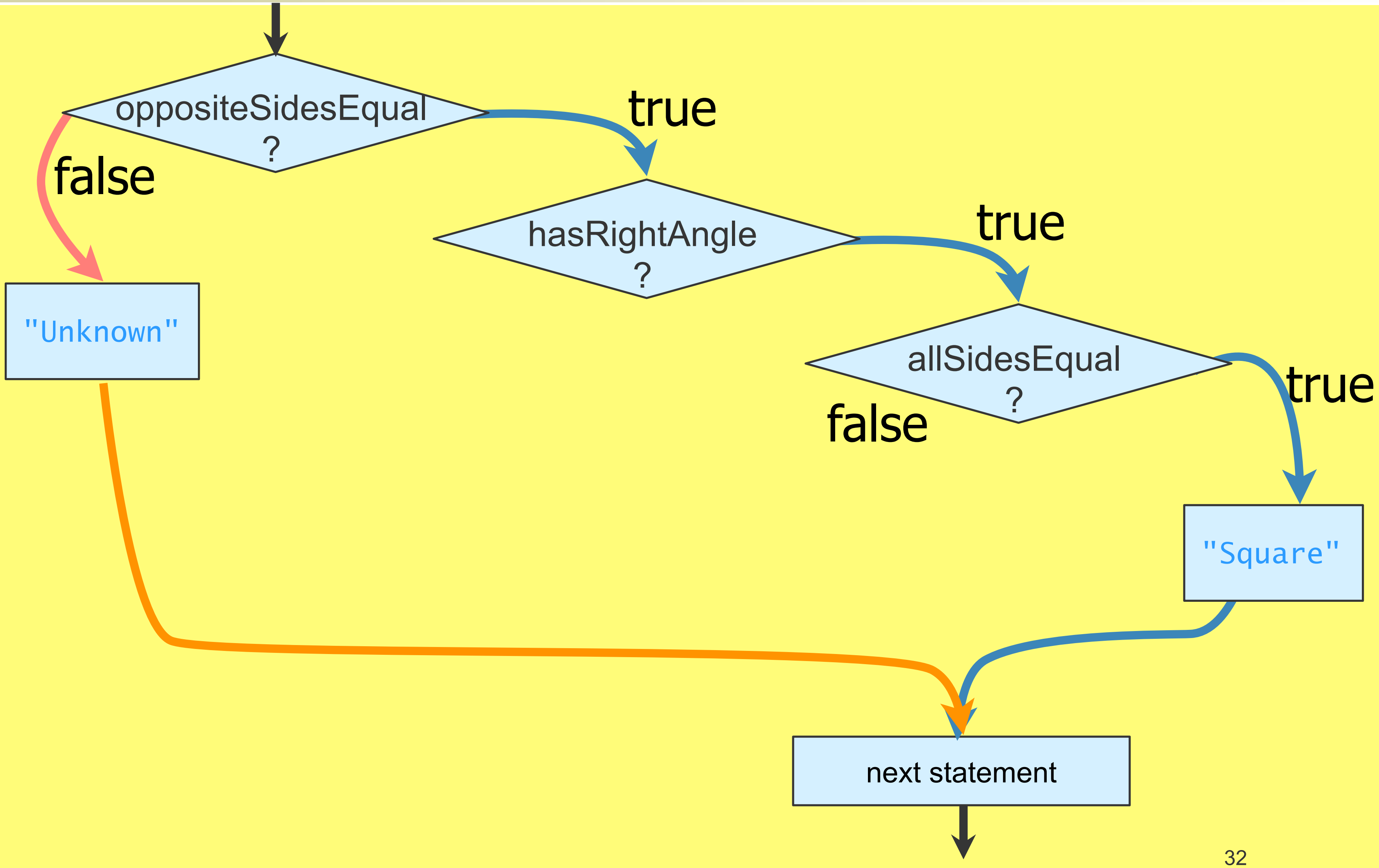
Quadrilateral Logic



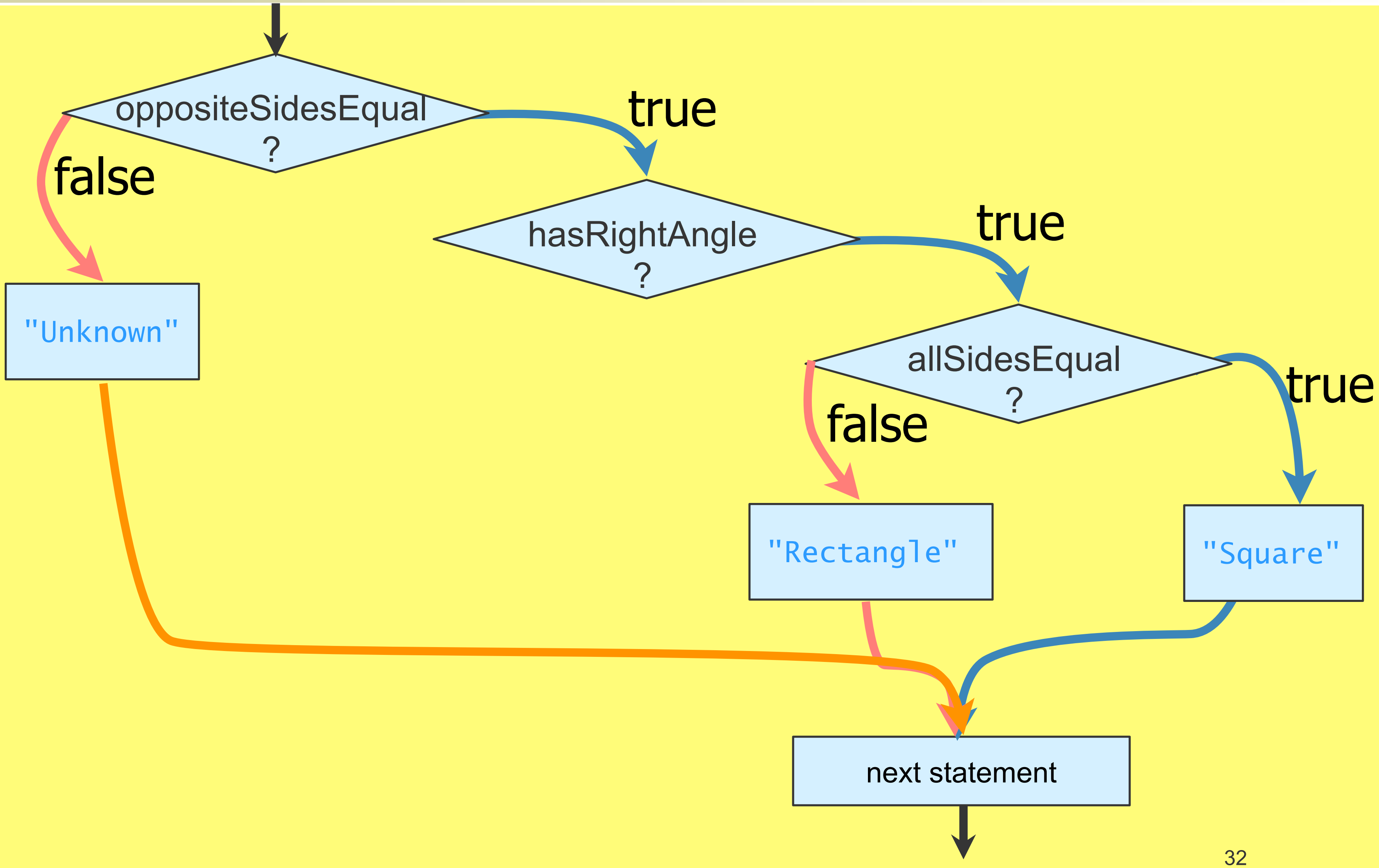
Quadrilateral Logic



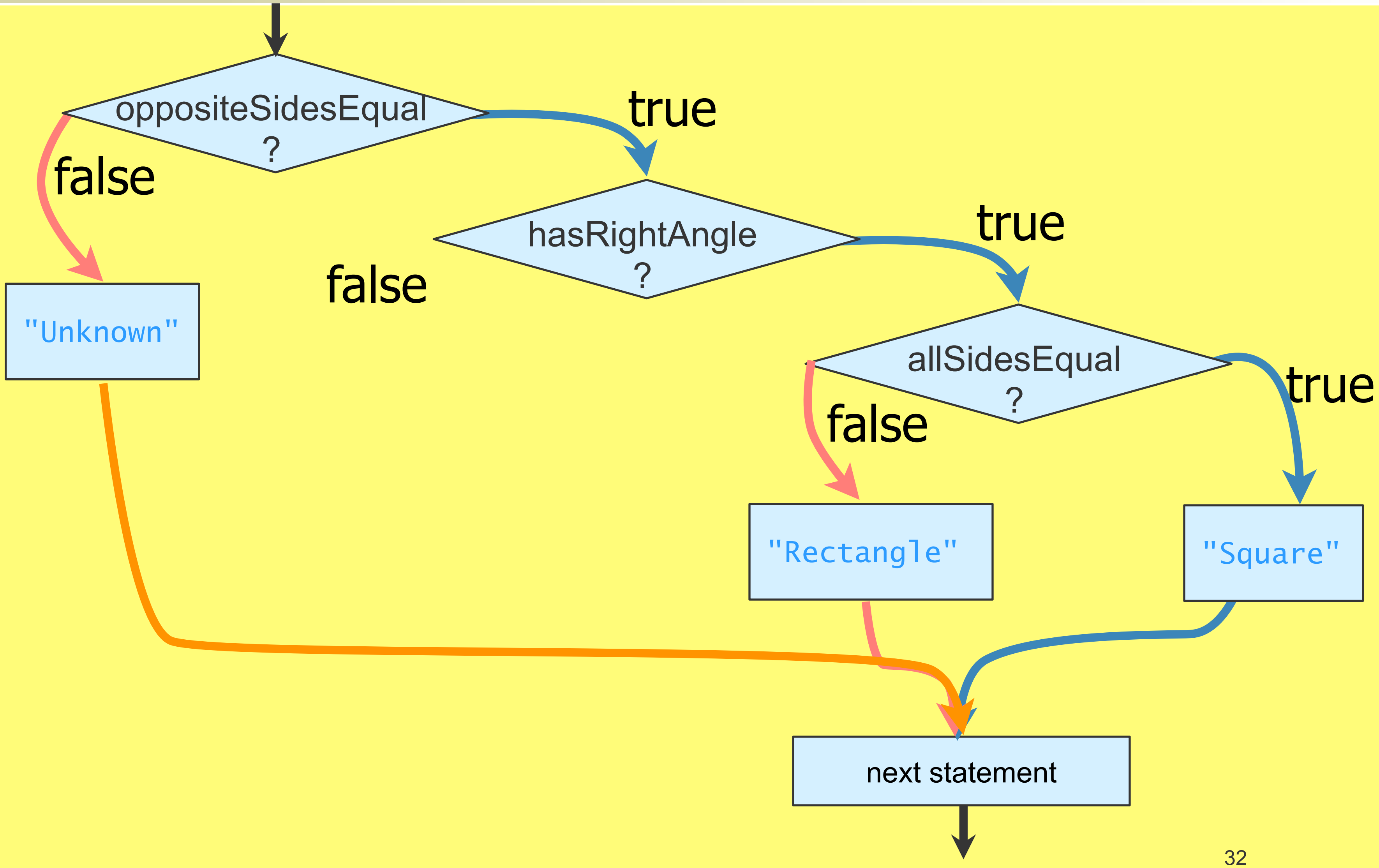
Quadrilateral Logic



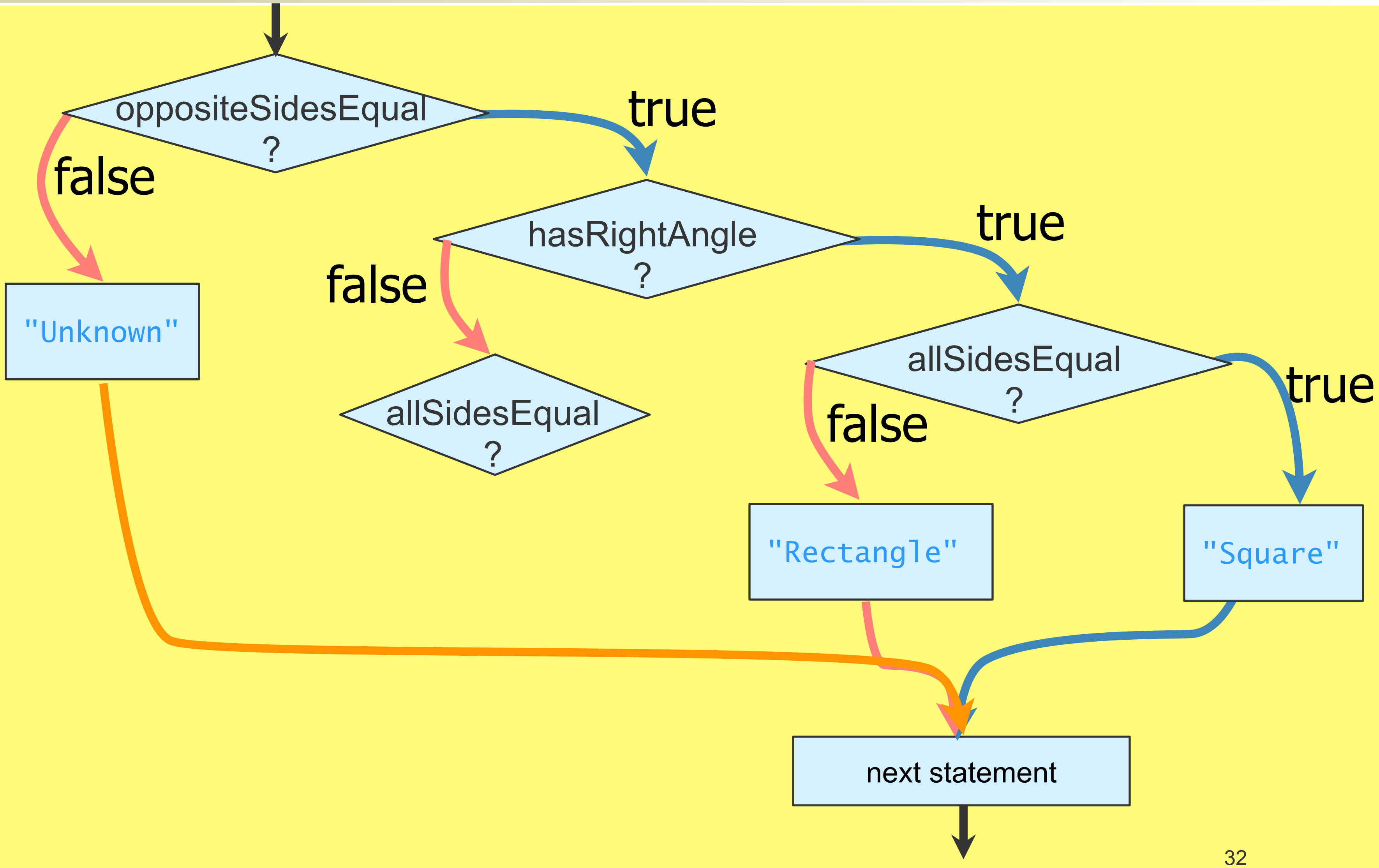
Quadrilateral Logic



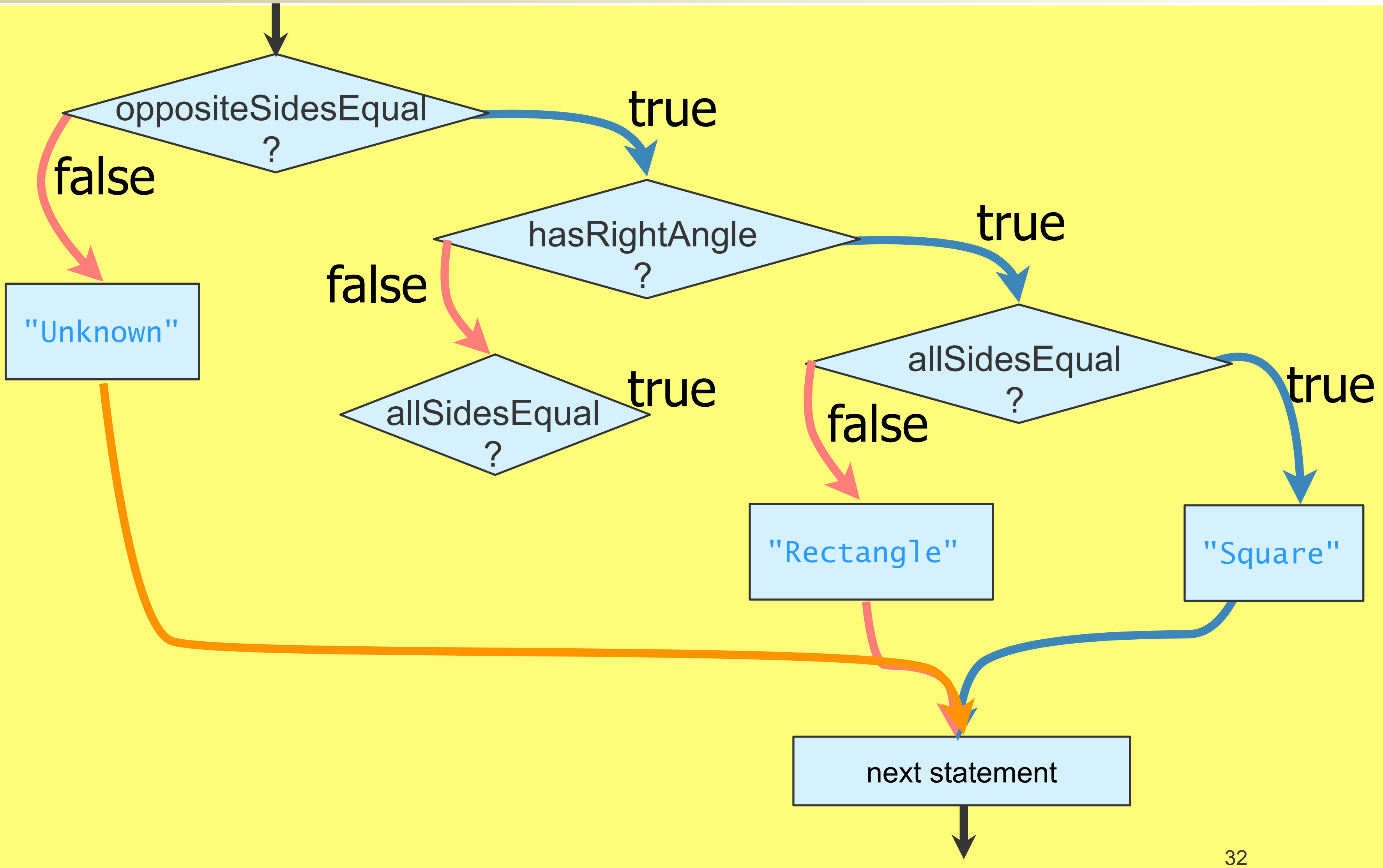
Quadrilateral Logic



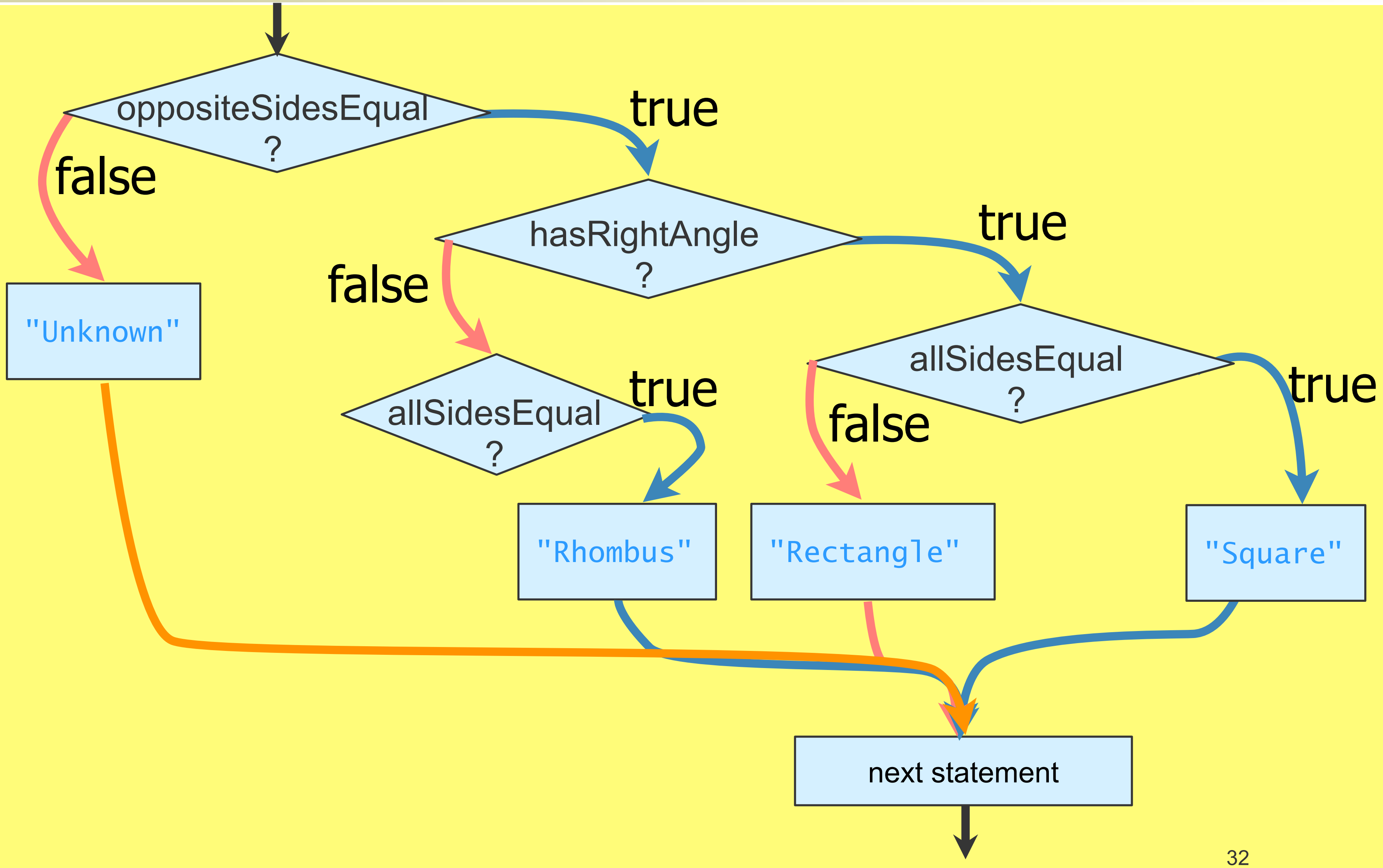
Quadrilateral Logic



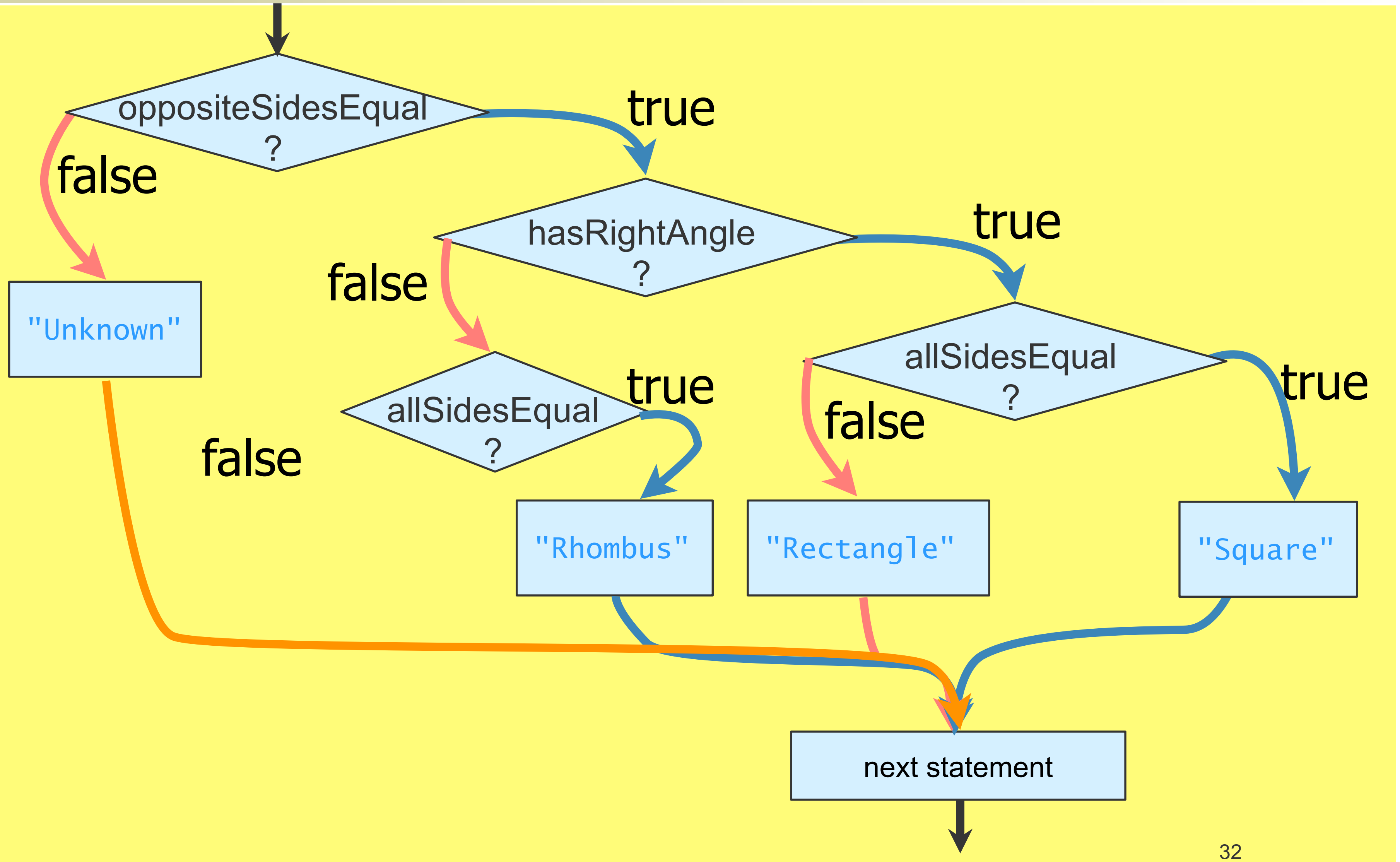
Quadrilateral Logic



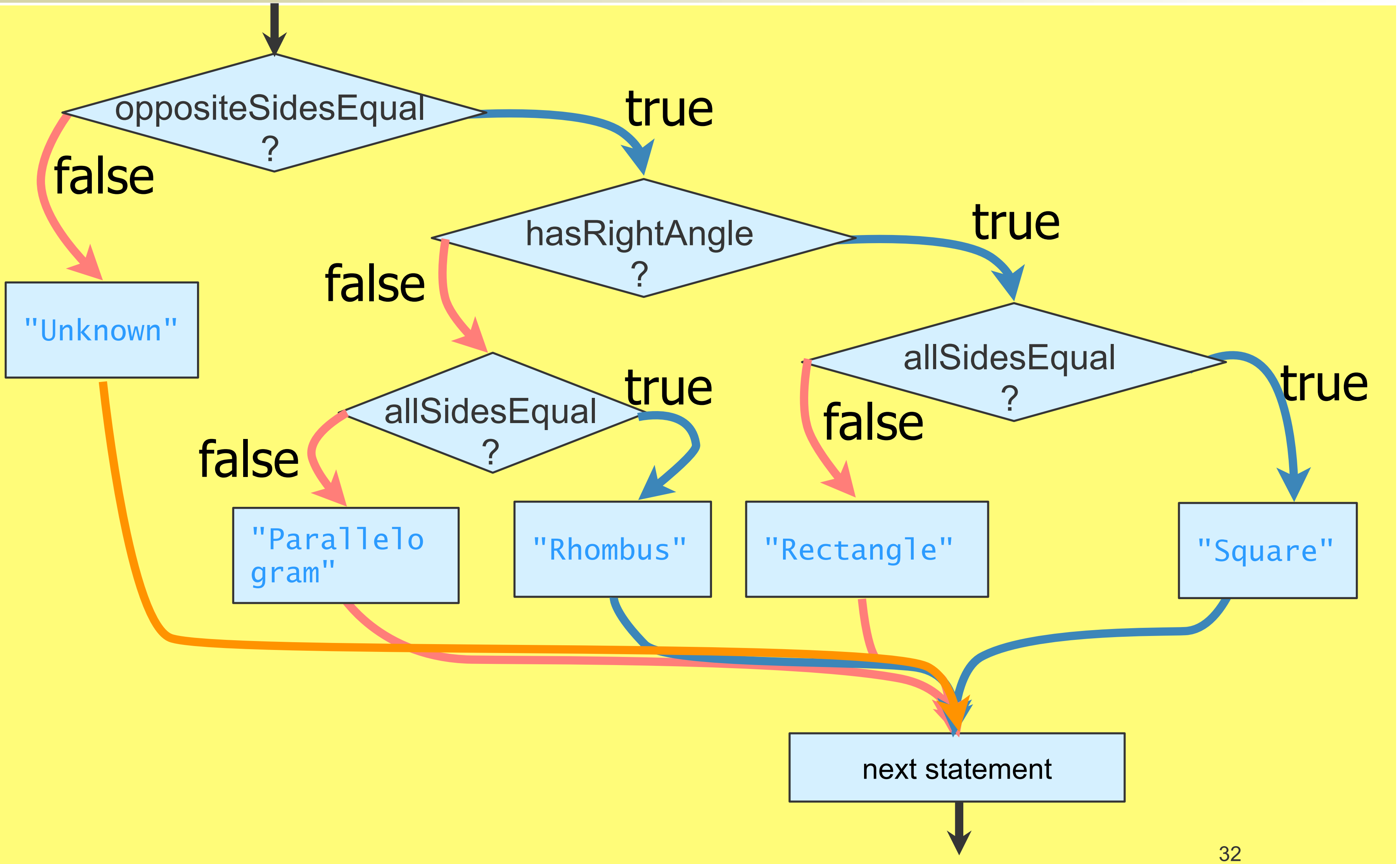
Quadrilateral Logic



Quadrilateral Logic



Quadrilateral Logic



```
public class QuadClassifier {
 public static void main(String args[]){
   int side1, side2, side3, side4;
   int anyAngle;
    . . . // read in all four side lengths and any one angle
   if ((side1==side3) && (side2==side4))
      if(anyAngle == 90)
         if(side1 == side2)
            type = "Square";
         else
            type = "Rectangle";
      else
         if(side1 == side2)
            type = "Rhombus";
         else
            type = "Parallelogram";
   else
      type = " type that is unfamiliar to this program";

   System.out.println("The quadilateral is a " + type);
```

QuadClassifier

```
public class QuadClassifier {
 public static void main(String args[]){
   int side1, side2, side3, side4;
   int anyAngle;
    . . . // read in all four side lengths and any one angle
   if ((side1==side3) && (side2==side4))
      if(anyAngle == 90)
         if(side1 == side2)
            type = "Square";
         else
            type = "Rectangle";
      else
         if(side1 == side2)
            type = "Rhombus";
         else
            type = "Parallelogram";
   else
      type = " type that is unfamiliar to this program";

   System.out.println("The quadilateral is a " + type);
```

QuadClassifier

```
public class QuadClassifier2 {
   . . .
   boolean oppositeSidesEqual, allSidesEqual, hasRightAngle;
    . . .
   oppositeSidesEqual = (side1==side3) && (side2==side4);
   allSidesEqual = oppositeSidesEqual && (side1 == side2);
   hasRightAngle = anyAngle==90;
   if (oppositeSidesEqual){
      if(hasRightAngle){
         if(allSidesEqual){
            type = "Square";
         } else {
            type = "Rectangle";
      } else {
         if(allSidesEqual) {
            type = "Rhombus";
         } else {
            type = "Parallelogram";
   } else {
      type = " type that is unfamiliar to this program";
   }

   System.out.println("The quadilateral is a " + type);
```

QuadClassifier2

```
public class QuadClassifier2 {
   . . .
   boolean oppositeSidesEqual, allSidesEqual, hasRightAngle;
    . . .
   oppositeSidesEqual = (side1==side3) && (side2==side4);
   allSidesEqual = oppositeSidesEqual && (side1 == side2);
   hasRightAngle = anyAngle==90;
   if (oppositeSidesEqual){
      if(hasRightAngle){
         if(allSidesEqual){
            type = "Square";
         } else {
            type = "Rectangle";
      } else {
         if(allSidesEqual) {
            type = "Rhombus";
         } else {
            type = "Parallelogram";
   } else {
      type = " type that is unfamiliar to this program";
   }

   System.out.println("The quadilateral is a " + type);
```

Easier to understand.

QuadClassifier2

Alternative styles

All are equivalent -- the compiler doesn't care.

Alternative styles

```
if ( <boolean expression> ) {
    …
}
else {
    …
}
```

All are equivalent -- the compiler doesn't care.

Alternative styles

```
if ( <boolean expression> ) {
    …
}
else {
    …
}
```

```
if ( <boolean expression> )
{
    …
}
else
{
    …
}
```

All are equivalent -- the compiler doesn't care.

Alternative styles

```
if ( <boolean expression> ) {
    …
}
else {
    …
}
```

```
if ( <boolean expression> ){
    …
} else {
    …
}
```

```
if ( <boolean expression> )
{
    …
}
else
{
    …
}
```

All are equivalent -- the compiler doesn't care.

Problem

- *Write a game program that requires the user to guess a random integer.*
- *After each input from the user*
    - *Let the user know if the guess was correct*
    - *Otherwise, inform the user that the guess was either too high or too low.*
- *The game ends only when the user correctly guesses the value.*

Repetition

- To solve this problem, we need the ability to repeat a set of operations (get input, compare with secret and respond) an unknown number of times
- The number is determined by how many guesses the user takes to get it right.
- This week we will learn how to repeatedly execute portions of code using **while**, and **do**-**while** loops.

Guess

```
public class Guess {
   public static void main(String[] args){
      int secret, guess;
      boolean done;
      Random random = new Random();
      secret = random.nextInt();
      done = false;
      while(!done){
         guess = Integer.parseInt(JOptionPane.showInputDialog(
                     null, "Enter your guess."));
         if(guess == secret){
            done = true;
            System.out.println("You guessed correctly!");
         } else if (guess < secret)
            System.out.println("Your guess was too low");
         else
            System.out.println("Your guess was too high");
      }
   }
}
```

Guess

Sentinel

```
public class Guess {
   public static void main(String[] args){
      int secret, guess;
      boolean done;
      Random random = new Random();
      secret = random.nextInt();
      done = false;
      while(!done){
         guess = Integer.parseInt(JOptionPane.showInputDialog(
                    null, "Enter your guess."));
         if(guess == secret){
            done = true;
            System.out.println("You guessed correctly!");
         } else if (guess < secret)
            System.out.println("Your guess was too low");
         else
            System.out.println("Your guess was too high");
      }
   }
}
```

Guess

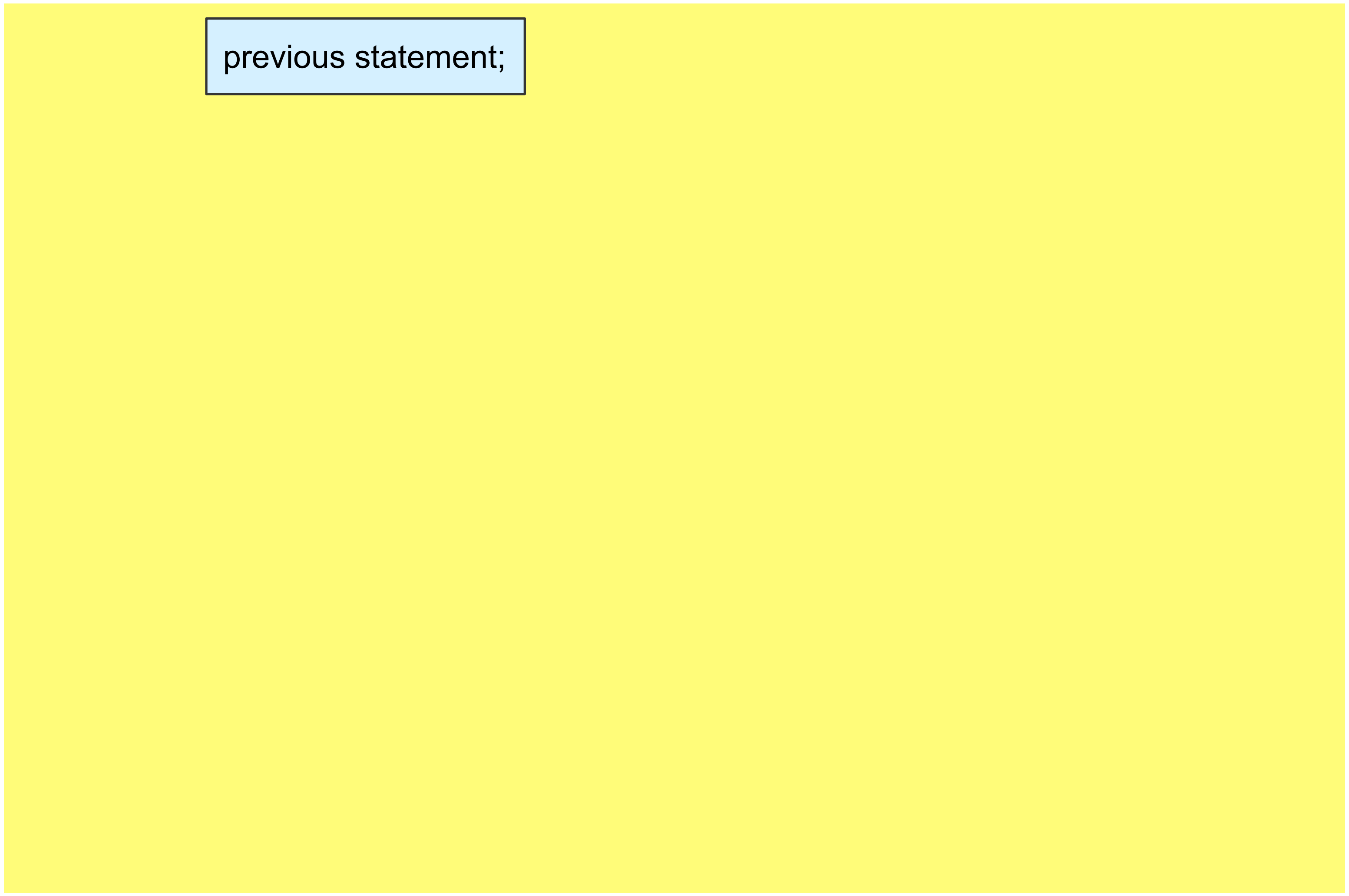
```
public class Guess {
   public static void main(String[] args){
      int secret, guess;
      boolean done;
      Random random = new Random();
      secret = random.nextInt();
      done = false;
      while(!done){
         guess = Integer.parseInt(JOptionPane.showInputDialog(
                     null, "Enter your guess."));
         if(guess == secret){
            done = true;
            System.out.println("You guessed correctly!");
         } else if (guess < secret)
            System.out.println("Your guess was too low");
         else
            System.out.println("Your guess was too high");
      }
   }
}
```

Sentinel

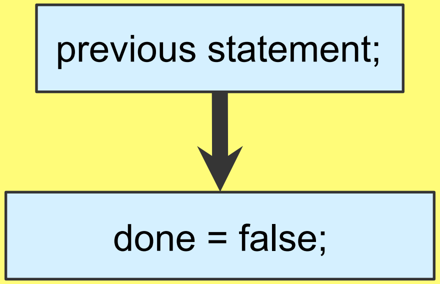
while loop

Control Flow of **while**

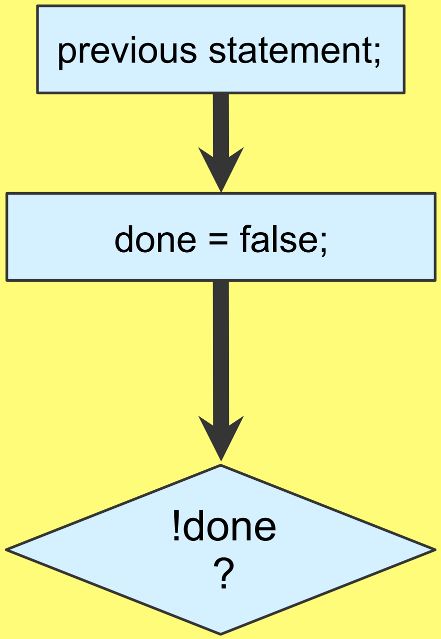
Control Flow of **while**



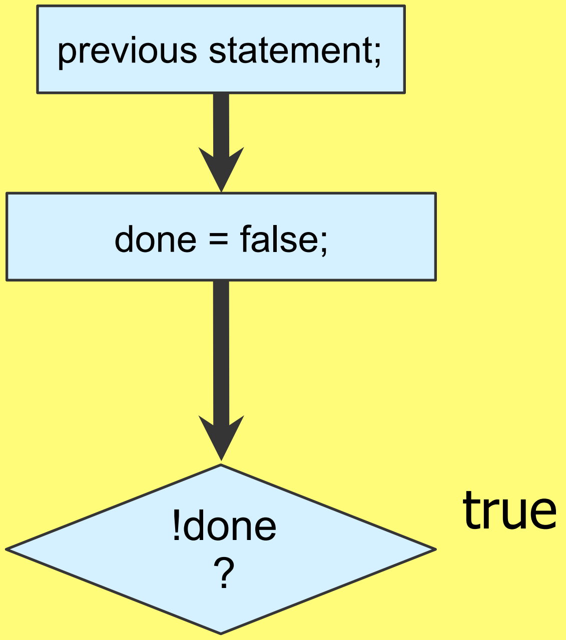
Control Flow of **while**



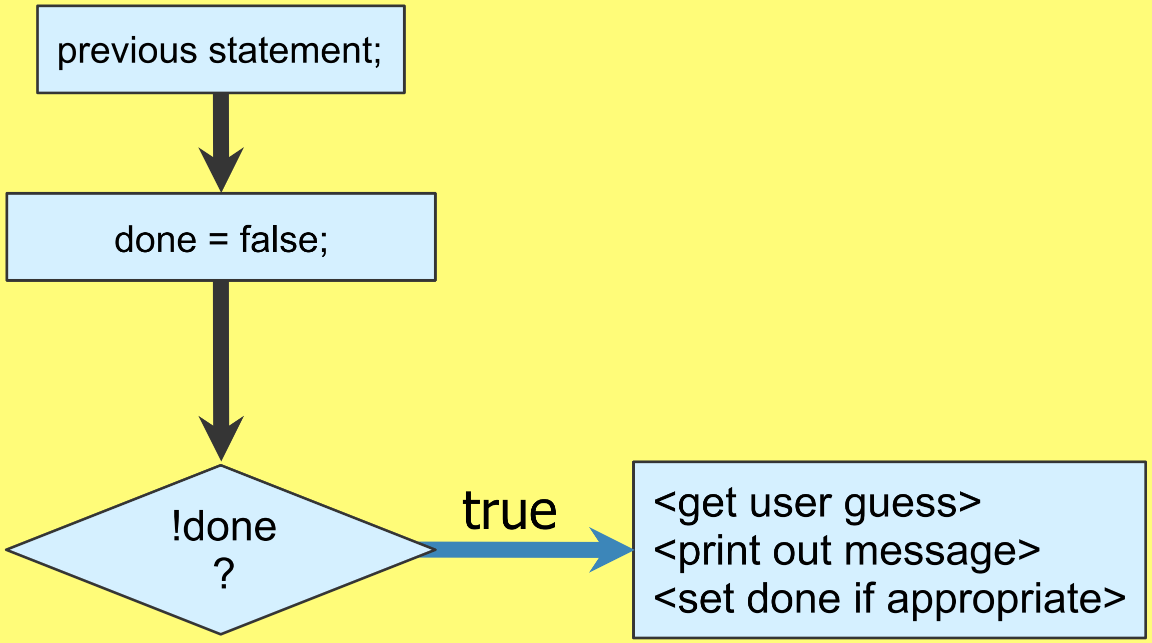
Control Flow of **while**



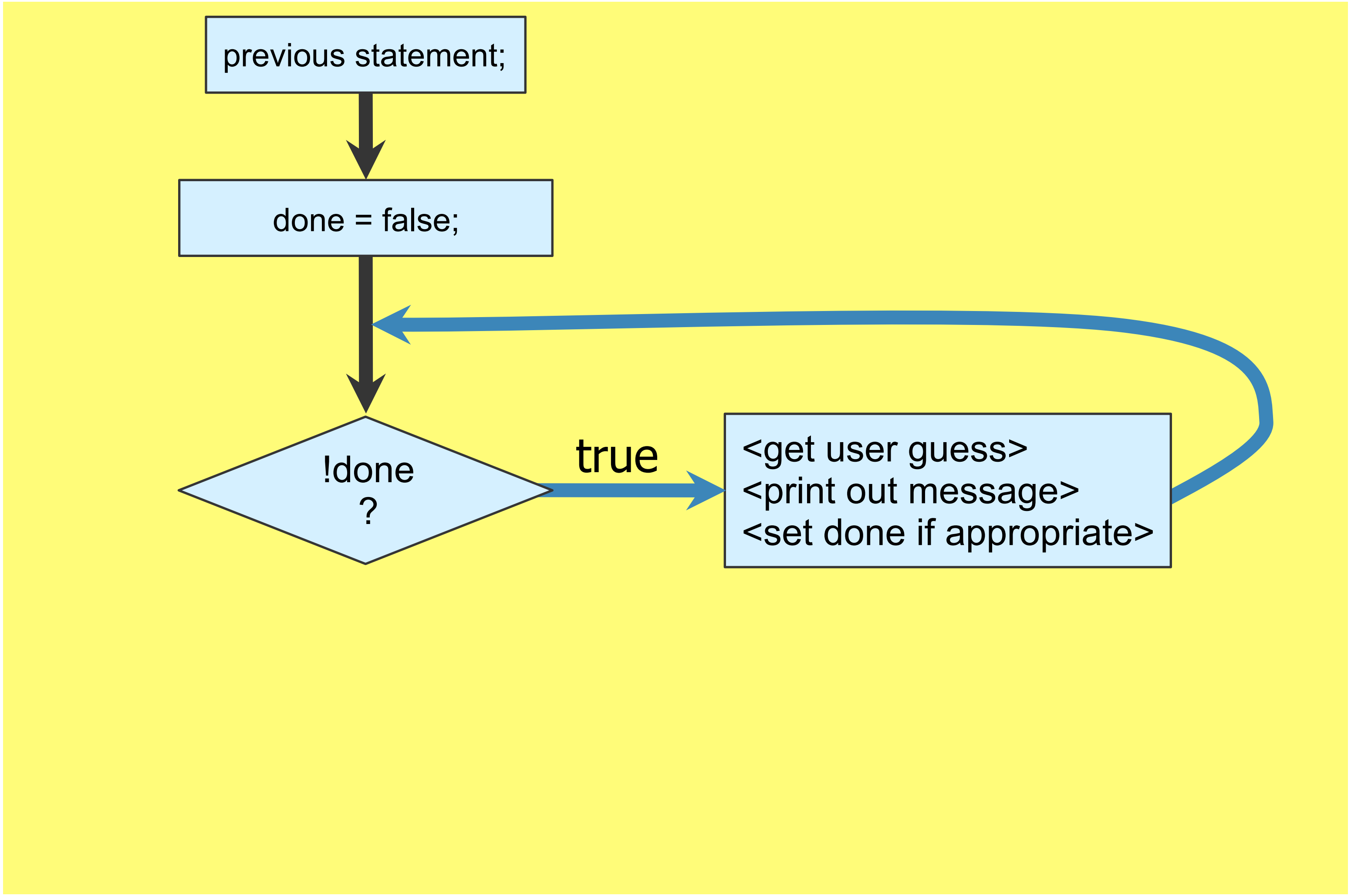
Control Flow of **while**



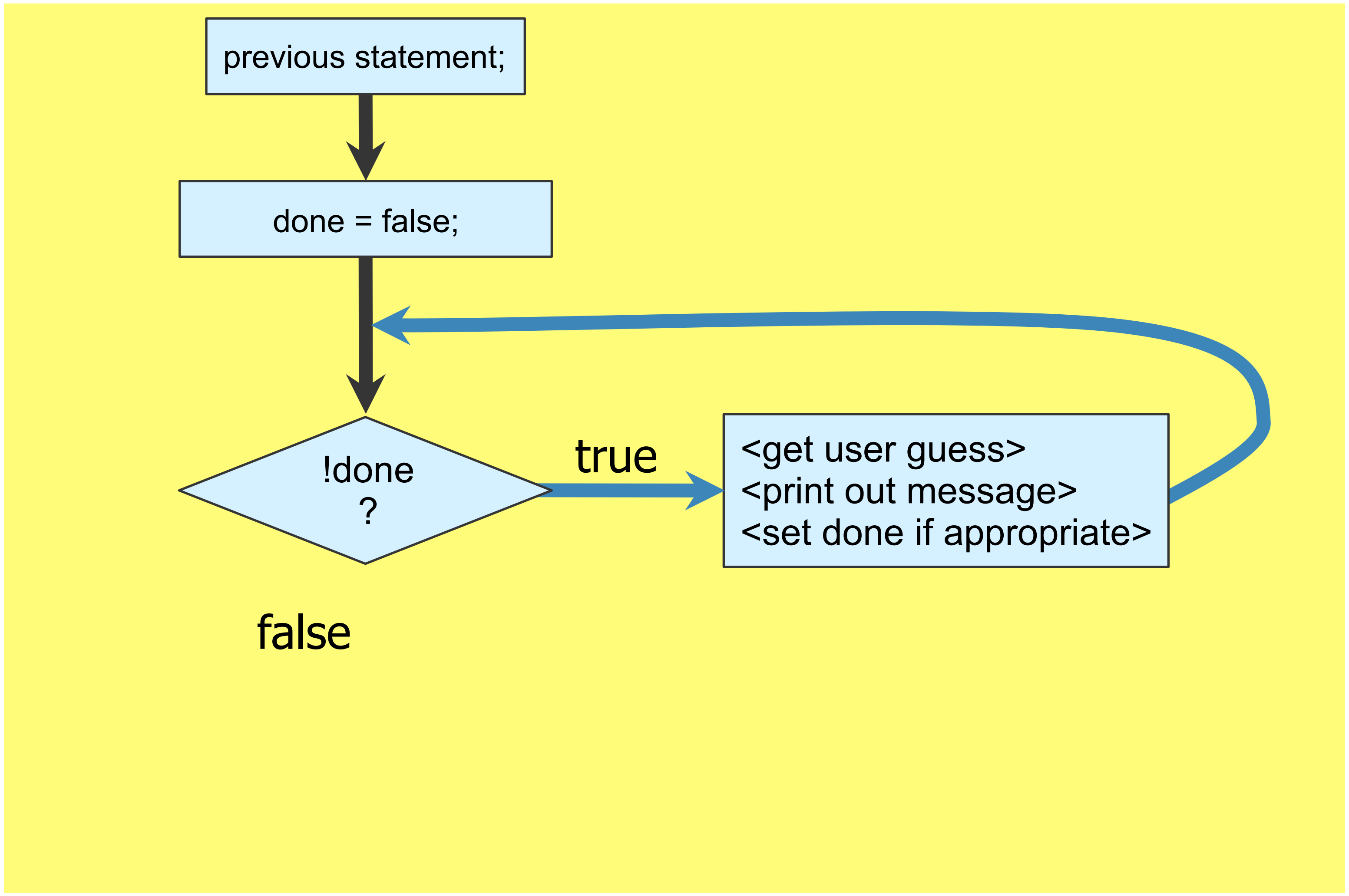
Control Flow of **while**



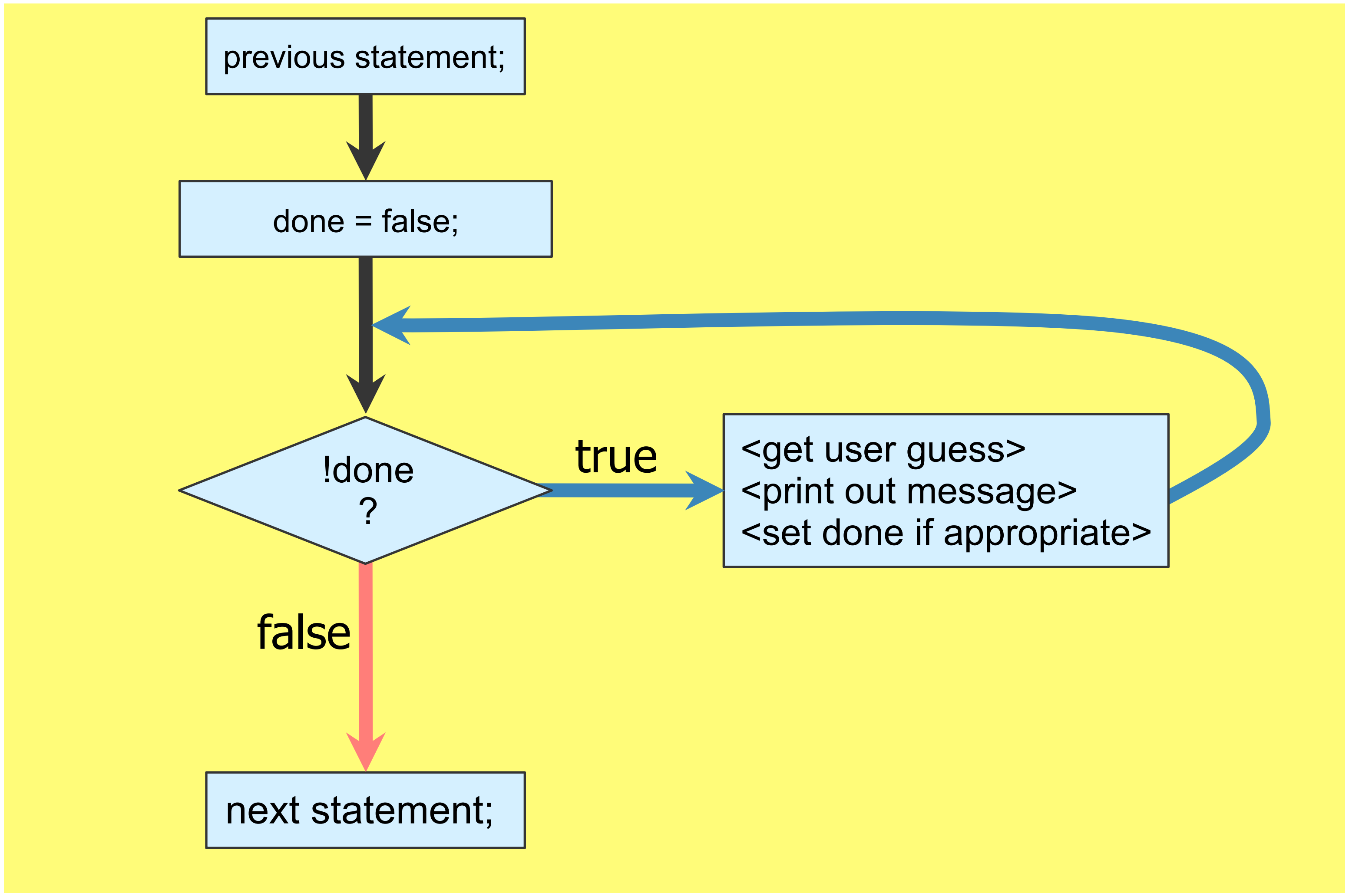
Control Flow of **while**



Control Flow of **while**



Control Flow of **while**



Syntax for the **while** Statement

```
while ( <boolean expression> )

        <statement>
```

```
while ( <boolean expression> ) {
        <statements>
}
```

Syntax for the **while** Statement

```
while ( <boolean expression> )

        <statement>
```

```
while ( <boolean expression> ) {
        <statements>
}
```

```
while ( !done )
{
    guess = Integer.parseInt(…);
    if (guess == secret)
      ...
}
```

Syntax for the **while** Statement

```
while ( <boolean expression> )

        <statement>
```

```
while ( <boolean expression> ) {
        <statements>
}
```

boolean expression

```
while ( !done )
{
    guess = Integer.parseInt(…);
    if (guess == secret)
      ...
}
```

Syntax for the while Statement

```
while ( <boolean expression> )

        <statement>
```

```
while ( <boolean expression> ) {
        <statements>
}
```

boolean expression

```
while ( !done )
{
    guess = Integer.parseInt(…);
    if (guess == secret)
      ...
}
```

loop body is repeatedly executed as long as boolean expression is **true**

PURDUE UNIVERSITY

Example: input check

```
char    credit;

credit = JOptionPane.showInputDialog(null, "Enter credit").charAt(0);

while (grade < 1 || grade > 5)
  credit = JOptionPane.showInputDialog(null,"Enter credit").charAt(0);
```

- Only accepts credits 1 through 5
- Note: need for initial input before loop
  - better option **do-while** loop

The do-while Statement

```
char     credit;
do {

  credit = JOptionPane.showInputDialog(null,"Enter credit").charAt(0);

} while ( credit < 1 || credit > 5 )
```

- Loop body executed before test (at least once).
- No need for initial input before loop

PURDUE
UNIVERSITY

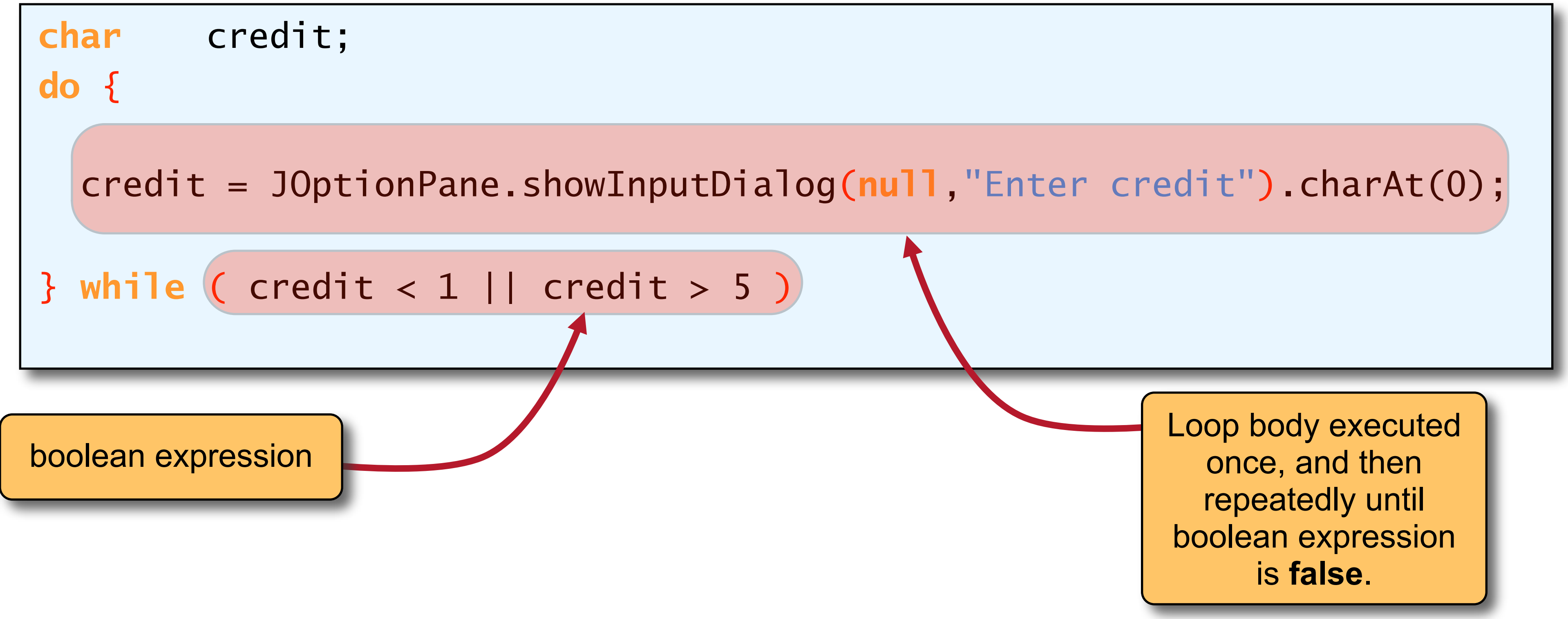
The do-while Statement

```
char     credit;
do {

  credit = JOptionPane.showInputDialog(null,"Enter credit").charAt(0);

} while ( credit < 1 || credit > 5 )
```

boolean expression

- Loop body executed before test (at least once).
- No need for initial input before loop

The do-while Statement



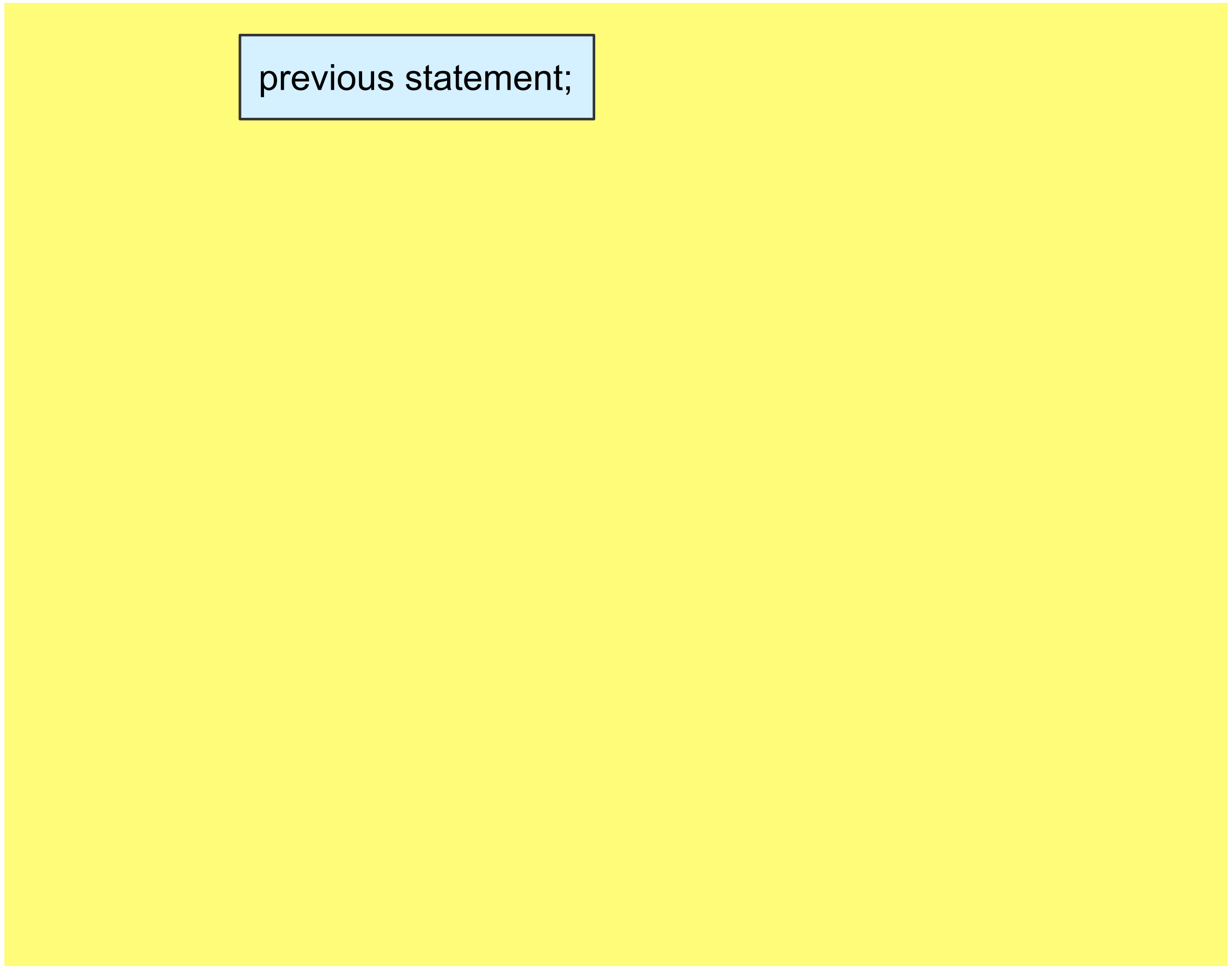
```
char     credit;
do {
  credit = JOptionPane.showInputDialog(null,"Enter credit").charAt(0);
} while ( credit < 1 || credit > 5 )
```

boolean expression
Loop body executed once, and then repeatedly until boolean expression is **false**.

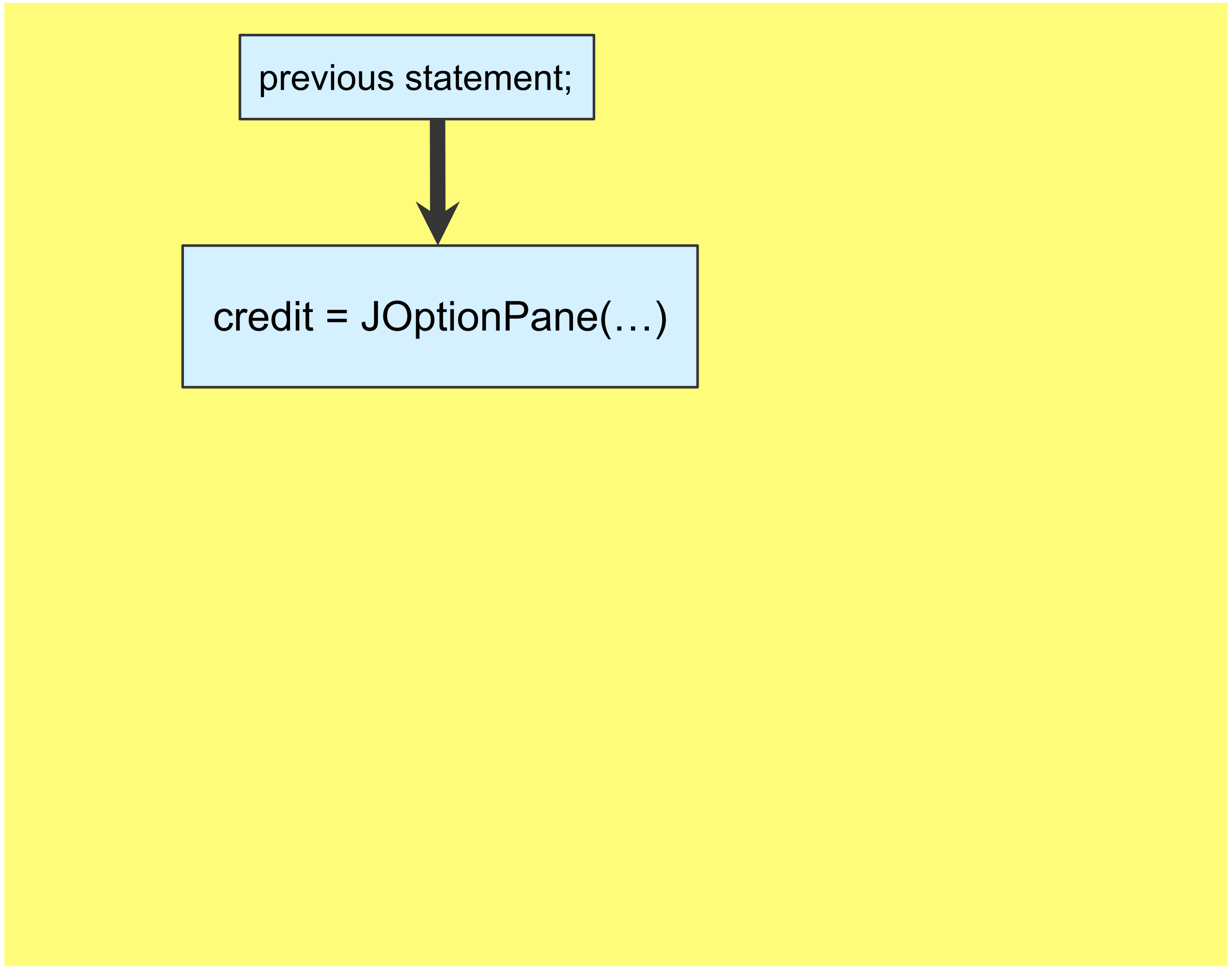
- Loop body executed before test (at least once).
- No need for initial input before loop

Control Flow of **do while**

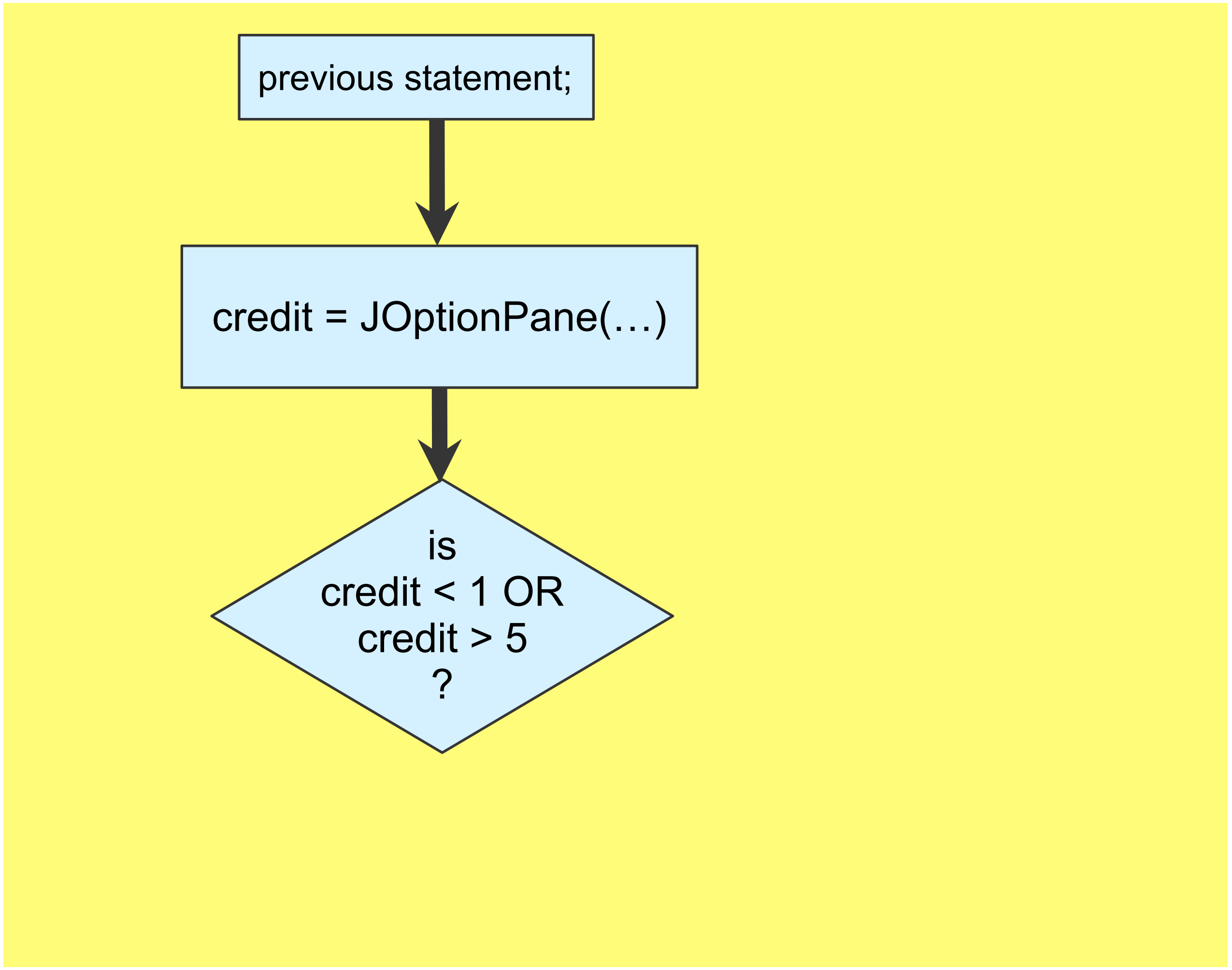
Control Flow of **do while**



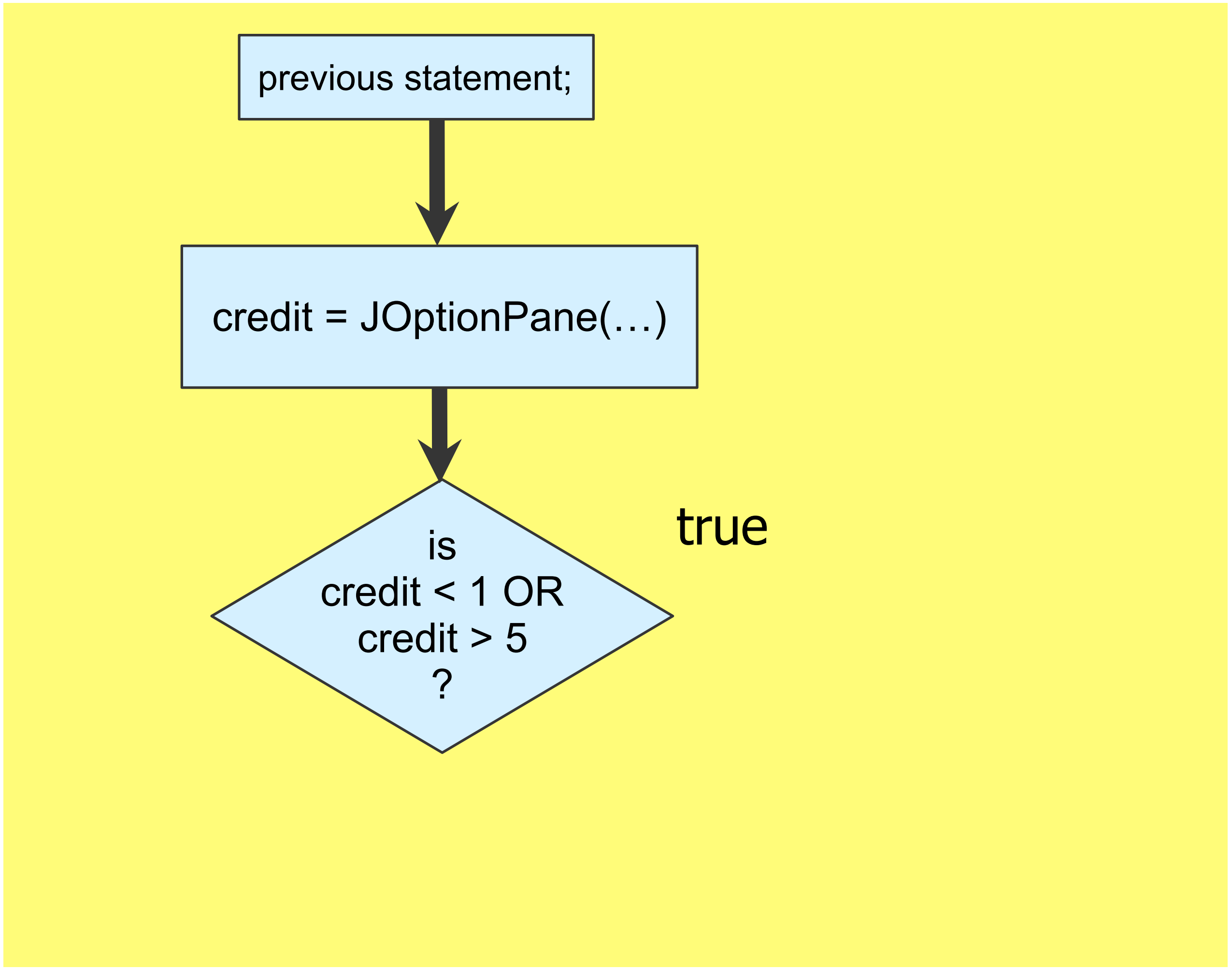
Control Flow of **do while**



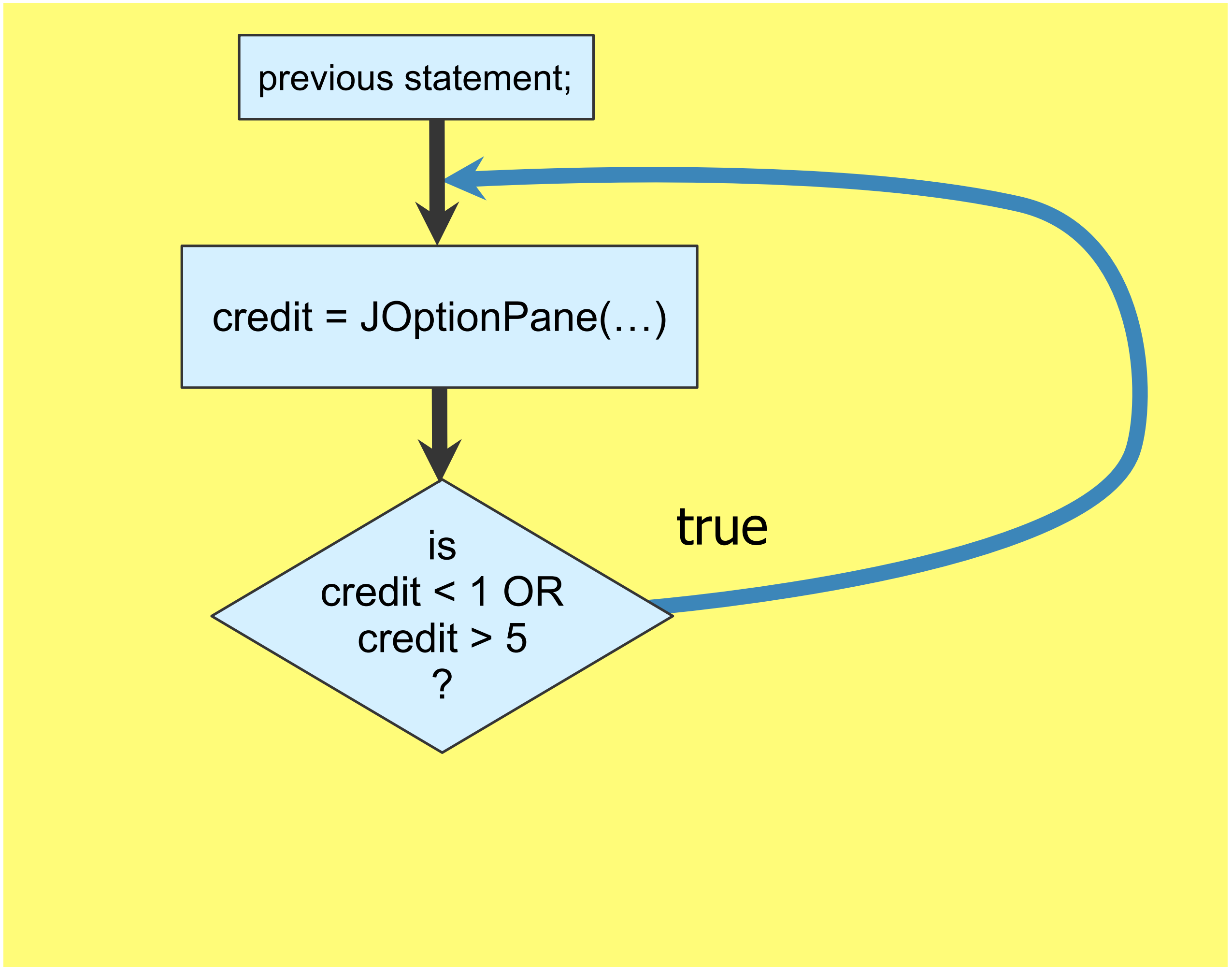
Control Flow of **do while**



Control Flow of **do while**

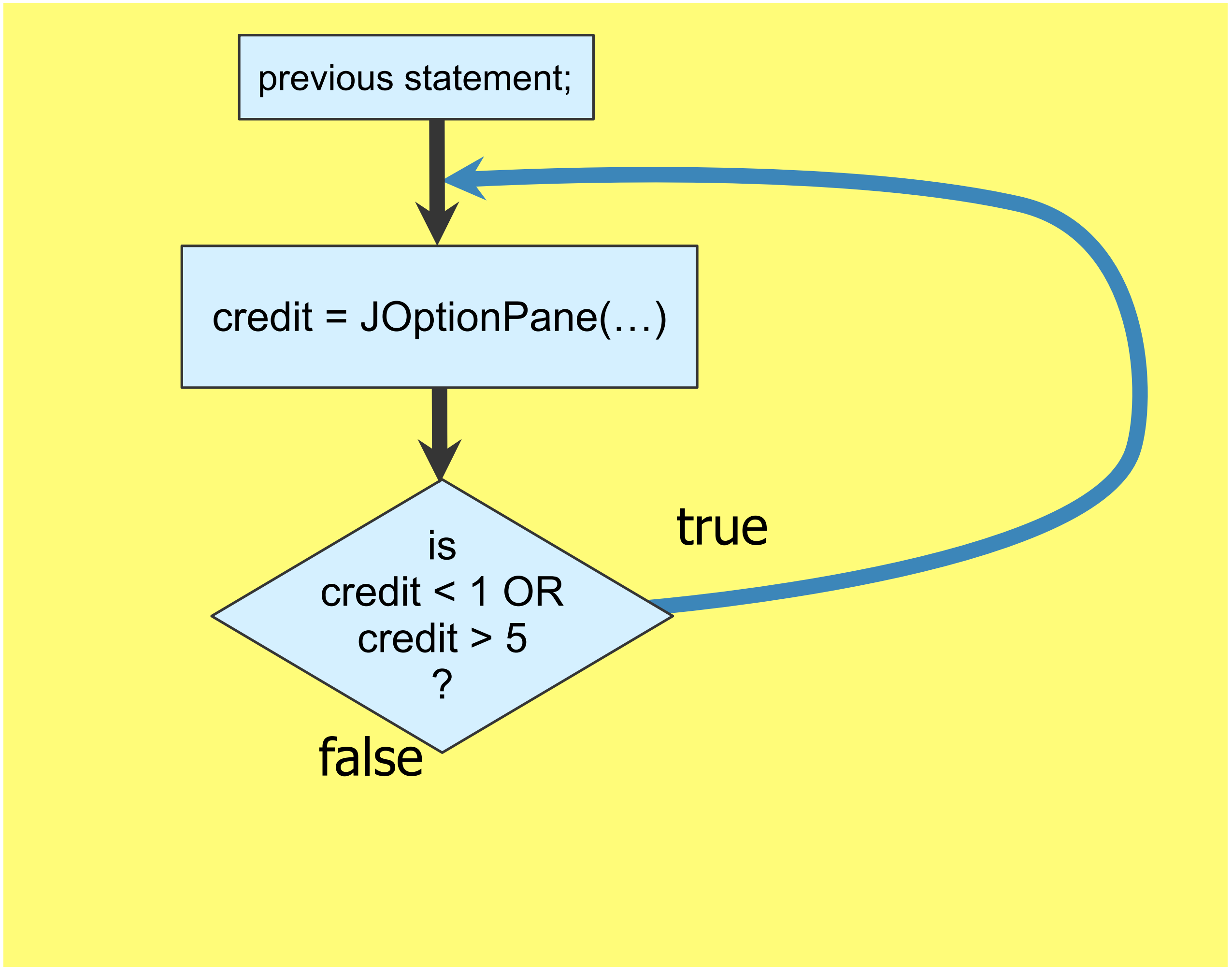


Control Flow of **do while**

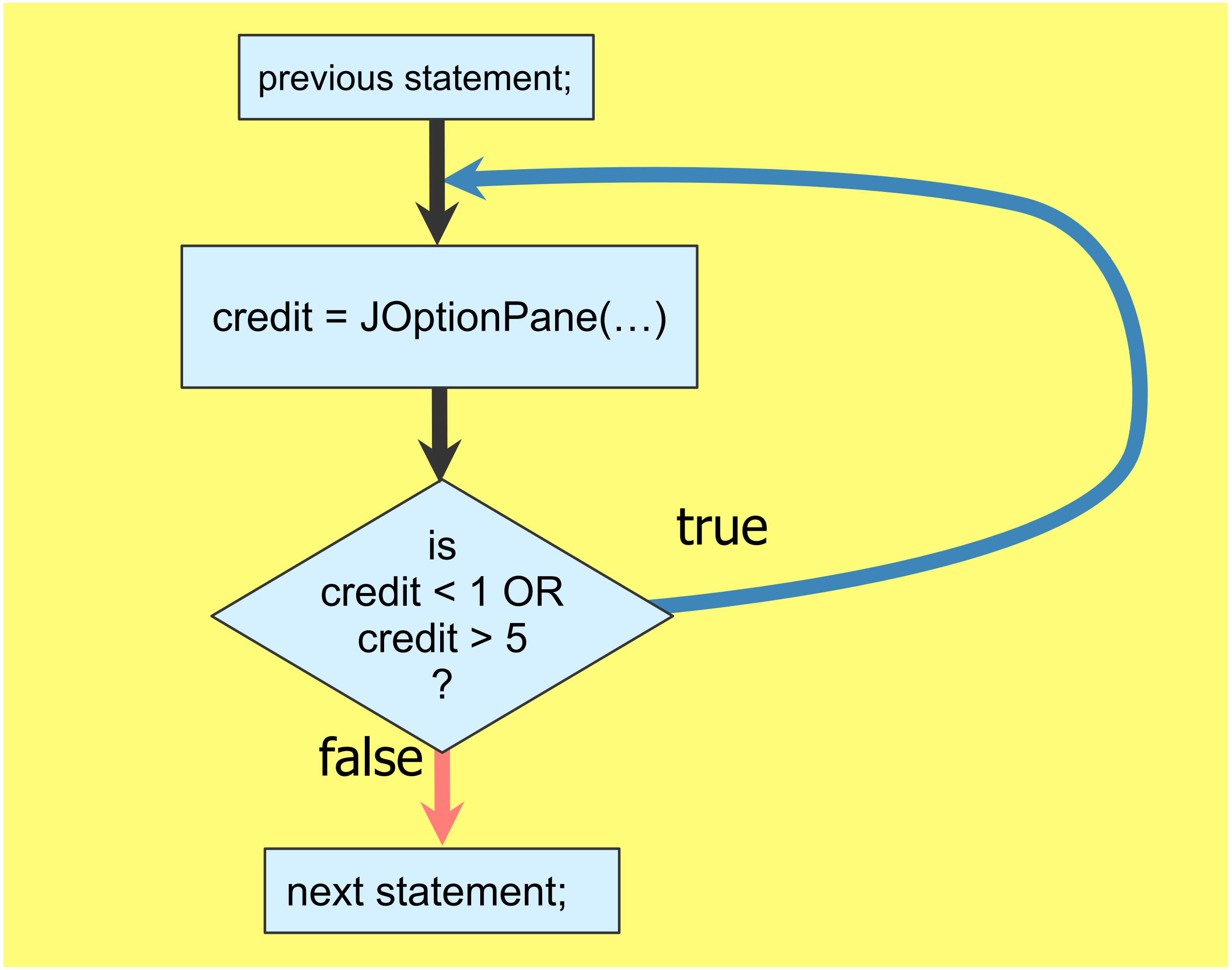


PURDUE UNIVERSITY

Control Flow of do while



Control Flow of **do while**



Common Errors

- Infinite loop
  - if the loop condition never becomes false the loop body will be executed endlessly
  - unless this is desired, ensure that the loop condition will change to false at some point

```
while(!done){
   guess = . . . ;
   if(guess == secret){
      done = true;
      System.out.println("You guessed
      correctly!");
   } else if . . .

}
```

Caution: Reals and Equality

1

```
float count = 0.0f;

while ( count != 1.0f ) {
    count = count + 0.3333333f;
}                //seven 3s
```

2

```
float count = 0.0f;

while ( count != 1.0f ) {
    count = count + 0.33333333f;
}                //eight 3s
```

PURDUE UNIVERSITY

Caution: Reals and Equality

1

```
float count = 0.0f;

while ( count != 1.0f ) {
    count = count + 0.3333333f;
}                  //seven 3s
```

2

```
float count = 0.0f;

while ( count != 1.0f ) {
    count = count + 0.33333333f;
}                  //eight 3s
```

Using Real Numbers
Loop 2 terminates, but Loop 1 does not because only an approximation of a real number can be stored in a computer's memory.

Loop Pitfall – 2a

1

```
int result = 0; double cnt = 1.0;
while (cnt <= 10.0){
    cnt += 1.0;
    result++;
}
System.out.println ( result);
```

2

```
int result = 0; double cnt = 0.111111111;
while ( cnt <= 1.11111111){
    cnt += 0.111111111;
    result++;
}
System.out.println ( result);
```

Loop Pitfall – 2a

1

```
int result = 0; double cnt = 1.0;
while (cnt <= 10.0){
    cnt += 1.0;
    result++;
}
System.out.println ( result);
```

→ 10

2

```
int result = 0; double cnt = 0.111111111;
while ( cnt <= 1.11111111){
    cnt += 0.111111111;
    result++;
}
System.out.println ( result);
```

→ 9

Using Real Numbers

Loop 1 prints out 10, as expected, but Loop 2 prints out 9. The value 0.111111111 cannot be stored precisely in computer memory.