Conditional Execution Using if

If you come to a fork in the road,

take it.

-- Yogi Berra

Conditional Statements

A *conditional statement* is one that *may* (or *may not*) be executed based on a *condition*.

Example:

if it is raining **then** I will study,

else I will go to the beach.

(it is raining) is the *condition*.

A *condition* is something that has a value of true or false (*boolean*).

Compound Conditional Statements

Conditional statements can be combined to form a *compound conditional statement.*

Example:

if it is raining **then** I will study,
else if it is cloudy **then** I will clean the yard,
else I will go to the beach.

Conditional Statement to Computer Code

Conditional statements are a key to writing useful computer programs. To express in computer code:

English:

if it is raining **then** I will study,
else I will go to the beach.

Program:

```
if ( is_raining ) study ;
else goToTheBeach ;
```

is_raining is a *boolean condition.*

study and goToTheBeach are *statements* or *actions.*

Syntax of a Conditional Statement

The Java (or C/C++/C#) syntax for a conditional statement is:

Syntax:

if (*test_condition*) *statement1* ;

else *statement2* ;

test_condition is anything that has a value of *true* or *false*

statement1 is the *action* to perform if the test is true.

statement2 is the *action* to perform if the test is false.

statement1 and *statement2* can be any legal statements.

NOTE: Java does not use the word "then".

How Do I Write A Test Condition?

To use conditional statements, you must know how to write a test condition. Here are a few examples. Details later.

Simple tests:

```
x > 0
choice == 1
scanner.hasNext( )    /* true if more input */
```

Compound tests:

```
x > 0 && x < 10      /* x > 0 and x < 10 */
choice == 1 || choice == 2   /* choice 1 or 2 */
```

Example Statements

If x is positive then add it to the score:

```
if ( x > 0 ) score = score + x;
```

If the score is more than 60, print "pass" else print "fail".

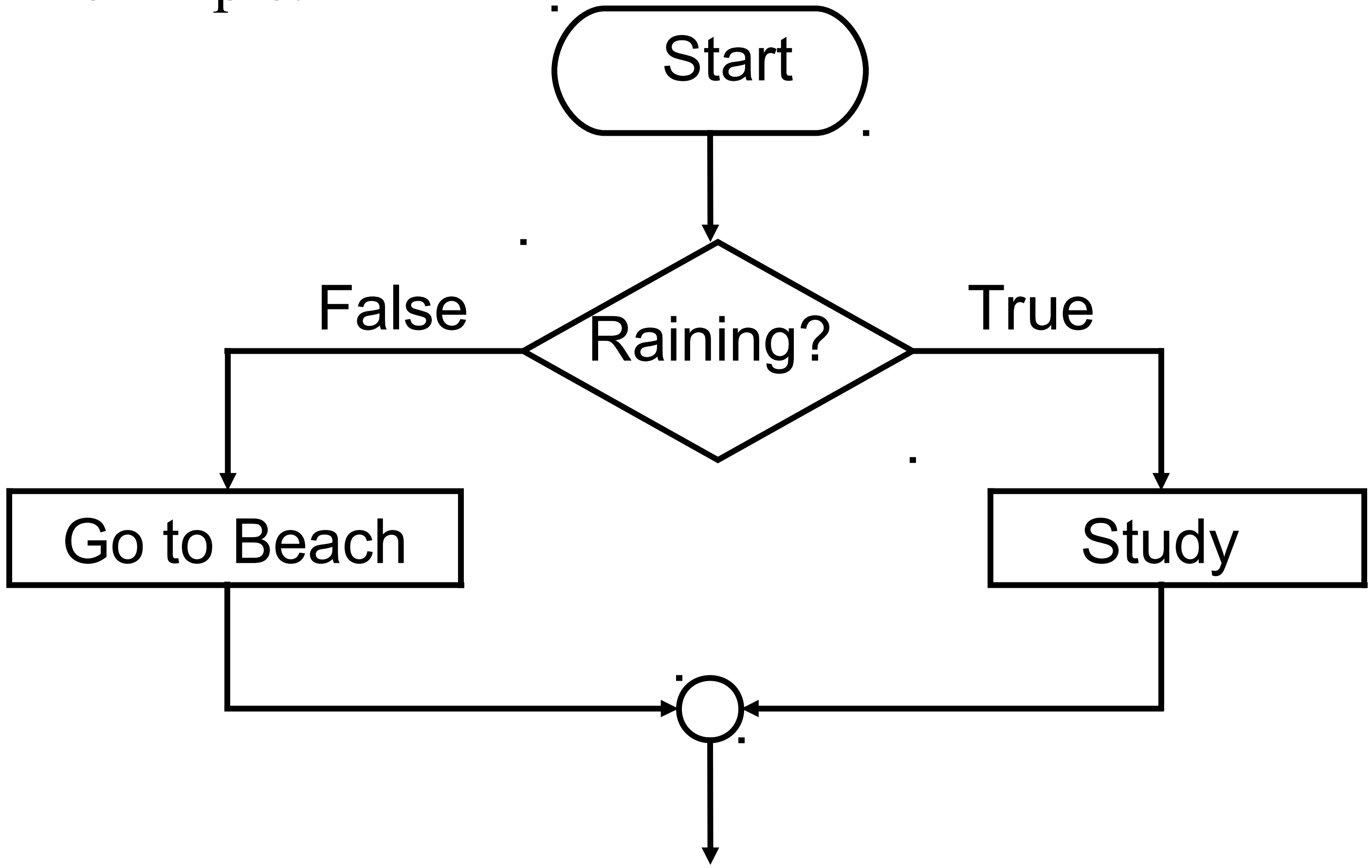
```
if ( score > 60 ) System.out.println( "pass” );

else System.out.println( "fail” );
```

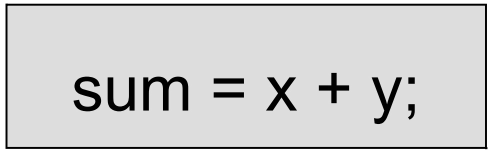
Must use semi-colons!

Flow Charts

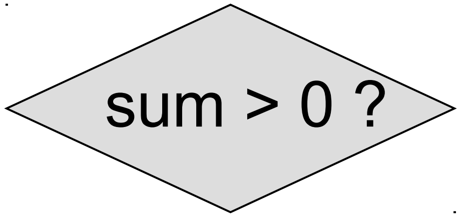
A *flow chart* can be useful to show conditional logic. Here's an example:



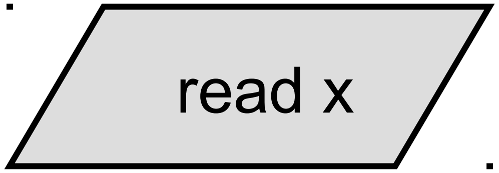
Flow Chart Symbols



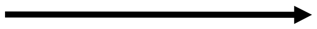
Process -- operations



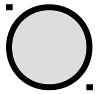
Condition



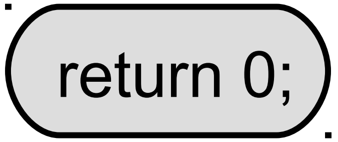
Input/Output



Flow line



Connector



Terminator

if … [then] … else ...

- if (*condition*) *statement;*   *// Java/C/Python do not*
- if (*condition*) *statement;*   *// use the word "then"*
  else *statement;*

1. "if" without any "else" clause:

```
if ( x > 0 ) sum += x;  // sum positive values
```

2. If x is positive then add to sum, else warn the user:

```
if ( x > 0 ) sum += x;  // sum positive values

else System.err.println("Sorry, x must be postive");
```

if With More Than One Action

An "if" statement can have more than one action:

English:

if it is raining **then** I will study,
    and then watch T.V.,
else I will go to the beach.

Program:

```
if ( is_raining ) {
    study;
    watchTV;
}
else goToTheBeach ;
```

Braces { ... } enclose a ***statement block***. You can use a *statement block* instead of a statement.

`Syntax of if` With Block

```
if ( n > 0 ) {

    statements to perform
    when n > 0

}
else {

    statements to perform
    when !(n > 0)

}
```

Multiple Action Example

English:

```
if score is positive then
    add score to the total
    increase count by 1
else
    display error message
```

Program:

```
if ( score > 0 ) {
    total = total + score; // add to the total score
    count++;        // add 1 to counter
}  else System.out.println( "invalid score: ”+score );
```

Compound Conditional Statements

A *compound conditional statement* has many branches.

English:

if it is raining **then** I will study,
else if it is cloudy **then** I will clean the yard,
else I will go to the beach.

Program:

```
if ( is_raining ) then study ;
else if ( is_cloudy ) then cleanTheYard ;
else goToTheBeach ;
```

Compound Conditional Example

A *compound conditional statement* has many branches.

English:

if score is more than 70 **then** pass,

else if score is more than 60 **then** try again,

else fail

Program:

```
if ( score > 70 ) System.out.println( "pass” );
else if ( score > 60 ) System.out.println( "try again” );
else System.out.println( "fail” );
```

Nested `if` Statement

```
% roll two dice
int die1 = rollDice( );  // = 1 ... 6
int die2 = rollDice( );  // = 1 ... 6
if ( die1 + die2 == 11 )
   System.out.println("You win!");
else
   if ( die1 == 6 )
      if ( die2 == 6 )
        System.out.println("Two 6es. Roll again.");
   else
      System.out.println("You lose.");
```

What will be output for each case?

```
Roll: 6  5      Output:
Roll: 6  6      Output:
Roll: 6  3      Output:
Roll: 3  6      Output:
```

Nested `if` Statement: *dangling else*

```
% roll two dice
int die1 = rollDice( );
int die2 = rollDice( );
if ( die1 + die2 == 11 )
   System.out.println("You win!");
else
   if ( die1 == 6 )
      if ( die2 == 6 )
         System.out.println("Two 6es. Roll again.");
   else
      System.out.println("You lose.");
```

An "else" clause pairs with the nearest unmatched "if" at the same block level.

```
Roll: 6  5        Output: You win!
Roll: 6  6        Output: Two 6es.  Roll again.
Roll: 6  3        Output: You lose.
Roll: 3  6        Output: (no output)
```

Avoiding dangling *else* confusion

- enclose the nested "if" in a { ... } block,

```
% roll two dice
int die1 = rollDice( );
int die2 = rollDice( );
if ( die1 + die2 == 11 )
   System.out.println("You win!");
else if ( die1 == 6 ) {
   if ( die2 == 6 )
        System.out.println("Two 6es. Roll again.");
   else
      System.out.println("You lose.");
}
```

This clarifies the logic,
but is not *really* what we want.

Avoiding dangling *else* confusion

- enclose nested "if" in a { ... } block, or
- structure the nested "if" as an if ... else if ... else .

```
% roll two dice
int die1 = rollDice( );
int die2 = rollDice( );
if ( die1 + die2 == 11 )
   System.out.println("You win!");
else if ( die1 == 6 && die2 == 6 )
   System.out.println("Two 6es. Roll again.");
else
   System.out.println("You lose.");
```

Much clearer -- every case has an action.

Relational operators

These relations return a value of true or false (boolean):

x == y	equality, must use 2 “=“ signs
x != y	not equal
x > y	greater than, greater than or equal
x >= y	greater than, greater than or equal
x < y	less than
x <= y	less than or equal

What is your grade if your total score is 90? 80? 79?

```
if ( total > 90 ) grade = "A";
else if ( total > 80 ) grade = "B";
else grade = "U"; // unsatisfactory
```

```
if ( total >= 90 ) grade = "A";
else if ( total >= 80 ) grade = "B";
else grade = "U";
```

Logical Operators and Compound Tests

expr1 && expr2 logical "and". expr2 is only evaluated if expr1 is true! (If expr1 is false, then the result is false.)

expr1 || expr2 logical "or". expr2 is only evaluated if expr1 is false! (If expr1 is true, then the result is true.)

! expr1 negate expr1. True if expr1 is false.

```
% comment on test score
if ( score > 90 ) comment = "excellent";
else if ( score > 70 && score <= 8 0 ) comment = "good";
else if ( score <= 70 ) comment =  "you party too much";
```

Compound Tests to Avoid Errors

```
if ( x/y < 0.1 ) System.out.println(“x/y is too small”);
```

What if y = 0 ? Division by zero will cause this program to fail. Solutions:

```
if ( y != 0)  if ( x/y < 0.1 ) System.out.println(“too small”);
```

Test y first. Test x/y only if y is not zero.

```
if ( y != 0 && x/y < 0.1 ) System.out.println(“too small”);
```

Same thing! Compiler knows that if first test is false, then the "and" condition is false. Skips second test.

True or False?

```
int n = 5, m = 10;
boolean answer1, answer2, answer3;
if ( n+m > 12 && n*m < 50 ) answer1 = true;
if ( n+m > 12 | | n*m < 50 ) answer2 = true;
if ( ! (n+m > 12 && n*m < 50) ) answer3 = true;
```

```
String s = new String( “Hello there” );
String t = "Hello " + "there";
boolean answer1 = ( s == t );
boolean answer2 = ( s < t );
boolean answer3 = s.equals( t );
```

(condition) ? expression1 : expression2

An inline version of "if … else ...".
The only ternary (3 argument) operator in Java. The usage is:

```
String grade;
grade = ( score > 60 ) ? “pass” : “fail”;
```

condition to test   do this if true   do this if false

```
// is the same as this…
if ( score > 60 ) grade = “pass”;
else grade = “fail” ;
```

Conditional Examples

```
// Compute quotient = numerator / denom.
// Avoid dividing by zero in case denom == 0
quotient = numerator / ( denom != 0 ) ? denom : 1 ;
```

```
// Announce new mail
int numMessages = getNewMail( );
System.out.println("You have " + numMessages
    + " new " +
  (numMessages == 1 ? "message" : "messages") );
```

You have 1 new message	if numMessages == 1
You have 3 new messages	any other value

Examples

Compound if ... else ...   (1)

Assign a grade using the variable score as follows:

grade = "A" if score >= 90

"B" if 80 <= score < 90

"C" if 65 <= score < 80

"D" if 50 <= score < 65

"F" if score < 50

```
int score = scanner.nextInt( );  // read score
String grade;
... write your code here ...
```

Compound if ... else ... (2)

Inefficient solution:

```
if ( score >= 90 ) grade = "A";
else if ( score >= 80 && score < 90 ) grade = "B";
else if ( score >= 65 && score < 80 ) grade = "C";
else if ( score >= 50 && score < 65 ) grade = "D";
else grade = "F";
```

Reason: duplicate tests waste time.

Compound if ... else ... (3)

Efficient solution:

```
if ( score >= 90 ) grade = "A";
else if ( score >= 80 ) grade = "B";
else if ( score >= 65 ) grade = "C";
else if ( score >= 50 ) grade = "D";
else grade = "F";
```

Reason: no duplicate tests.

"if" succeeds quickly for cases with score > 80, avoiding many tests.

Compound if ... else ...   (4)

Efficient solution for a *bad class*:

```
if ( score < 50 ) grade = "F";
else if ( score < 65 ) grade = "D";
else if ( score < 80 ) grade = "C";
else if ( score < 90 ) grade = "B";
else grade = "A";
```

This is efficient if you a *bad class* (most scores < 65), because it will succeed for bad scores first. If you have a *good class* (most scores >= 80) then the previous slide is more efficient.

Early return from a method (1)

In a program, this task would probably be placed in a method.

```
private String computeGrade( int score ) {
    String grade;
    if ( score >= 90 ) grade = "A";
    else if ( score >= 80 ) grade = "B";
    else if ( score >= 65 ) grade = "C";
    else if ( score >= 50 ) grade = "D";
    else grade = "F";
    return grade;
}
```

Q: Can you write without using a compound "if" and "grade"?

Early return from a method (2)

Return from the method as soon as grade is known:

```
private String computeGrade( int score ) {
    if ( score >= 90 ) return  "A";
    else if ( score >= 80 ) return  "B";
    else if ( score >= 65 ) return  "C";
    else if ( score >= 50 ) return  "D";
    else return  "F";
}
```

That eliminates useless assignment to local variable "grade".
Can you eliminate the compound "if" statement?

Early return from a method (3)

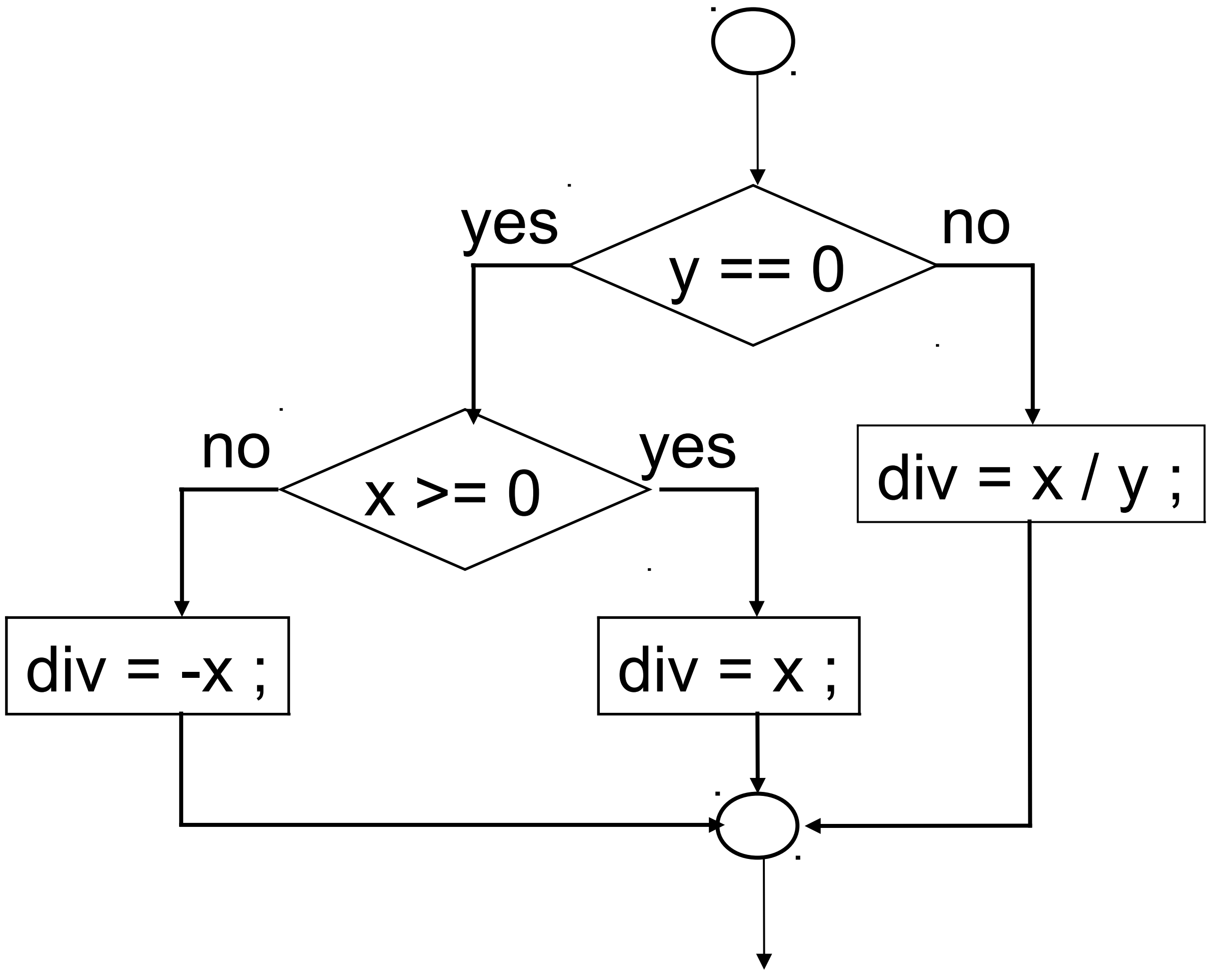
Previous side is the same as this:

```
private String computeGrade( int score ) {
    if ( score >= 90 ) return  "A";
    if ( score >= 80 ) return  "B";
    if ( score >= 65 ) return  "C";
    if ( score >= 50 ) return  "D";
    return  "F";
}
```

A compiler will usually produce the same code as in the previous slide, so use whichever form you like best.

(I like the previous one because it shows logical structure; some people like this form for simplicity.)

Construct Conditional from Flow Chart

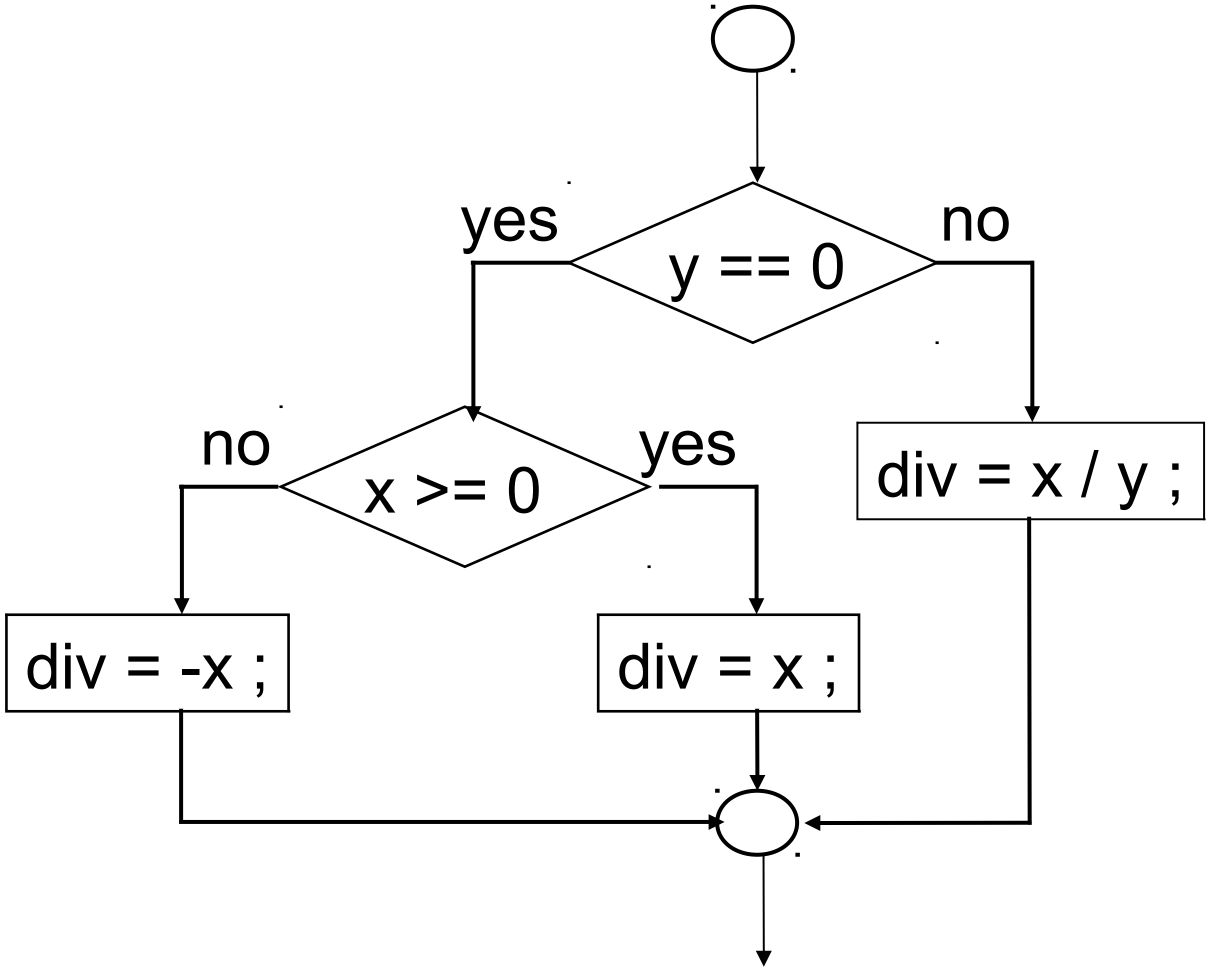


write Java code

to implement this

flow chart

Construct Conditional from Flow Chart



```
if ( y != 0 ) div = x/y;
else if ( x >= 0 )
    div = x;
else div = -x;
```