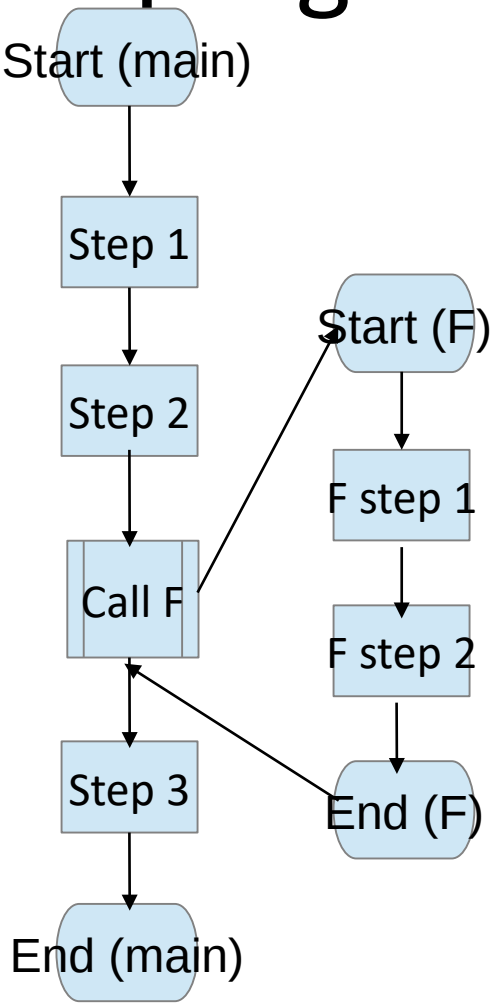
Flow of control in a program

- Program starts with the first statement in main
- Instructions proceed sequentially:
  - One at a time
  - Top to bottom
  - Each executes exactly once
- If a method call occurs, the sequence is interrupted; when the method returns, the sequence resumes



Altering flow of control

- Programming constructs can be employed to alter the flow of logic
- Two basic categories:
  - Selection (aka branching)
  - Iteration (aka looping)
- Both constructs rely on the evaluation of logical or relational expressions to determine which instruction(s) to execute

Quick review of logical expressions

- Logical, or boolean expressions, are expressions that evaluate to true or false
- Relational operators:

  >, <, >=, <=

  ==, !=
- Logical operators:

  &&

  ||

  !

Selection structures in Java

- Simplest structure is plain if:

```
if(expression evaluates true)
{
    // perform instruction(s)
}
```

- Example:

```
if ((x % 10) == 0)
{
    System.out.printf("%d is a multiple of 10\n", x);
}
```

Selection structures in Java

- Important syntax notes:
  - ***Always*** put parentheses around the expression to be tested in an if clause (it's a syntax error if you don't)
  - ***Never*** put a semicolon at the end of the if clause
    - Semicolon, all by itself, is a null (non-operational) statement
    - if (expression); says if expression is true, do nothing
  - Begin/end brackets ({}) are required if more than one statement is to be executed because of an if clause; if there is only one statement, they are optional

The if/else structure: syntax

```
if (expression)
{
    // statement(s) to perform if expression is true
}
else
{
    // statement(s) to perform if expression is false
}
```

Compound selection (multiple branching)

- Each if and else clause is followed by one or more statements to execute
- We can place another if or if/else combination within the set of statements followed by either clause
- The result is a ***nested*** if or nested if/else structure

Notes on nested if/else

- An else clause ***never*** has its own condition; it is always the alternative to the nearest previous if clause
- If you want to test another condition, therefore, you need another if clause – this was the case with the previous program:

```
if(r > 0)
    System.out.printf("%d is positive\n", r);
else if (r < 0) /* new condition, new if clause */
```

Matching ifs and elses

- An else clause always pairs with the closest if clause that doesn't already have a paired else
- Example

```
if (A)
        ...
        if (B)
                ...
                if (C)
                        ...
                else /* pairs with condition C */
                        ...
        else /* pairs with condition B */
                ...
else /* pairs with condition A */
        ...
```

Alternate branching structures: switch/case & the ternary operator

- Nested if/else structures are sufficient to cover any program you might want to write that involves logical selection
- There are a couple of alternative structures you should at least be aware of
- These are briefly discussed on the next 2 slides

The switch/case structure

- May be used instead of if/else when the condition being tested is in the form “if x == y”
- Instead of using the relational expression, we use the variable (x) as the “switch,” and the various possible values (y) as “cases”
- Instead of a final else, we can use a default case

The ternary operator

- Is so named because it takes 3 operands
- Syntax:

  expression ? statement1 : statement2
- Logic:
  - If the expression is true, perform statement1
  - Otherwise, perform statement 2