CS 241 — Introduction to Problem Solving and Programming

Fundamentals of Programming

Flow of control

Jan 21, 2005

Overview

- The `boolean` type
- Branch statements
- Boolean operators

Boolean

We've mentioned another primitive type besides `int`, `double`, and `char`: the type `boolean`.

- Named after logician/mathematician George Boole
- Used to represent values in boolean logic
- Has two values: `true` and `false`.

Boolean type

We can declare and use variables, use the literals, and print them out.

```
boolean x = true;
boolean y = false;
System.out.println("x: " + x + ", y: " + y);
```

```
...

x: true, y: false
```

Boolean operators

We can produce values using boolean-valued operators.

```
        int first = DocsIO.readint("Please enter first number-->");
        int second = DocsIO.readint("Please enter second number-->";

        boolean lt = first < second;
        boolean lteq = first <= second;
        boolean eq = first == second;
        boolean gteq = first >= second;
        boolean gt = first > second;

        System.out.println(lt + " / " + lteq + " / " + eq
                           + " / " + gteq + " / " + gt);
```

```
...

Please enter first number-->5;
Please enter second number-->7
true / true / false / false / false
```

Boolean operators

Here are the boolean-valued operators that operate on `ints`, `doubles`, and related types.

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

Note the difference between = (assignment) and == (comparison).

Boolean operators

What if we mixed up assignment and comparison?

```
boolean firstEqualsSecond = first = second;
boolean firstEqualsTwelve = 12 = first;

...

BoolBasic.java:17: incompatible types
found   : int
required: boolean
       boolean firstEqualsSecond = first = second;
                                   ^
BoolBasic.java:18: unexpected type
required: variable
found   : value
       boolean firstEqualsTwelve = 12 = first;
                                   ^
2 errors
```

Boolean operators

For characters, comparison is in lexicographical order.

```
char first = 'A';
char second = 'A';
char third = 'Z';
char fourth = 'z';

System.out.println((first == second) + " / " + (first == third));
System.out.println((first < second) + " / " +
                   (second < third) + " / " +
                   (third < fourth));
```

```
...

true / false
false / true / true
```

Use of boolean

Booleans are used for making decisions.

Suppose we wanted to print a remainer only if it is not zero.

```
int first, second, quotient, remainder;
first = DocsIO.readint("Enter dividend: ");
second = DocsIO.readint("Enter divisor: ");
quotient = first /  second;
remainder = first % second;
String result = first + " / " + second + " = " + quotient;
if (remainder != 0)
   result += " R" + remainder;
System.out.println(result);
```

Use of boolean

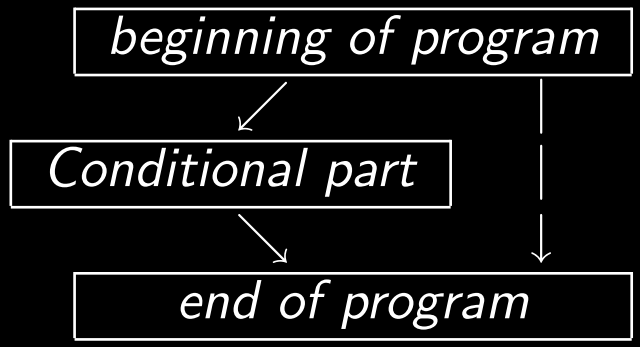
```
Enter dividend: 23
Enter divisor: 4
23 / 4 = 5 R3

...

Enter dividend: 24
Enter divisor: 4
24 / 4 = 6
```

Branch statement

Such a decision-making expression is a branch statement or an if statement.

This aspect of the program is flow of control.

Branch Statement: `if` (*BooleanExpression*)
Statement



Block statements

What if we wanted more than one thing to happen under a condition?

```
int first, second, quotient;
first = 23;
second = 4;
quotient = first /  second;
System.out.println(first + " / " + second + " = " + quotient);
if (first != second * quotient) {
   int remainder = first % second;
   System.out.println("The remainder is " + remainder);
}
```

Block statements

Curly braces set off a block statement.

```
int first, second, quotient;
first = 23;
second = 4;
quotient = first /  second;
System.out.println(first + " / " + second + " = " + quotient);
if (first != second * quotient) {
   int remainder = first % second;
   System.out.println("The remainder is " + remainder);
}
```

Block statements

Variables declared in a block are live only in the block.

```
int first, second, quotient;
first = 23;
second = 4;
quotient = first /  second;
System.out.println(first + " / " + second + " = " + quotient);
if (first != second * quotient) {
   int remainder = first % second;
   System.out.println("The remainder is " + remainder);
}
```

This is called the variable's scope.

Variable scope

```
        System.out.println(first + " / " + second + " = " + quotient);
        if (first != second * quotient) {
           int remainder = first % second;
           System.out.println("The remainder is " + remainder);
        }
        System.out.println(quotient + " R " + remainder);

...

Quotient.java:14: cannot resolve symbol
symbol  : variable remainder
location: class Quotient
        System.out.println(quotient + " R " + remainder);
                                              ^
1 error
```

Block statement

Block Statement: {

Statement
Statement
Statement
Statement . . .

}

Zero or more statements enclosed in curly braces.

These will prove fundamental to many constructs we'll see later. . .

If / else

What if there is more than one alternate action?

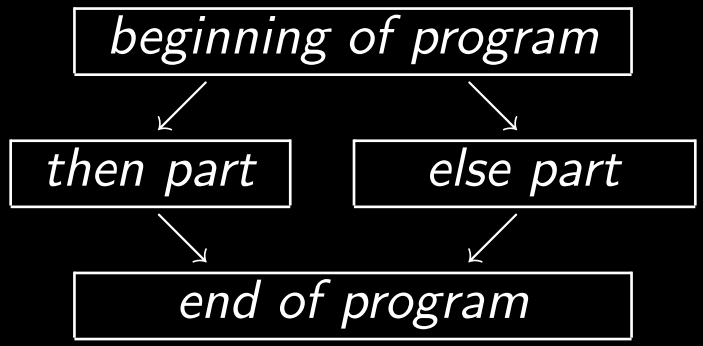
Use `else`.

```
int first, second, quotient, remainder;
first = 23;
second = 4;
quotient = first /  second;
remainder = first % second;
if (first != second * quotient)
    System.out.println(first + " / " + second + " = "
                       + quotient + " R " + remainder);
else
    System.out.println(first + " / " + second + " = " + quotient);
```

If /else

The `else` branch is an optional part of a branch statement.

Branch Statement: `if` (*BooleanExpression*)
Statement
`else`
Statement



If / else

```
int guess = DocsIO.readint("Please guess a number, 1 to 99-->");
if (guess == 16)
   System.out.println("That is correct!");
else
   System.out.println("I'm sorry, " + guess + " is wrong.");
```

```
...

Please guess a number, 1 to 99-->53
I'm sorry, 53 is wrong.

...

Please guess a number, 1 to 99-->16
That is correct!
```

If / else

What if there are several alternatives?

We could always nest branch statements in block statements.

```
if (guess == 16)
   System.out.println("That is correct!");
else {
   if (guess < 1)
      System.out.println("That's not even in the range.");
   else {
      if (guess > 99)
         System.out.println("That's not even in the range.");
      else
         System.out.println("I'm sorry, " + guess + " is wrong.");
   }
}
```

If / else

This block is actually a single branch statement.

We don't need the curly braces.

```
if (guess == 16)
   System.out.println("That is correct!");
else {
   if (guess < 1)
      System.out.println("That's not even in the range.");
   else {
      if (guess > 99)
         System.out.println("That's not even in the range.");
      else
         System.out.println("I'm sorry, " + guess + " is wrong.");
   }
}
```

If / else

This block is also a single statement.

We'll get rid of the braces here, too.

```
if (guess == 16)
   System.out.println("That is correct!");
else {
   if (guess < 1)
      System.out.println("That's not even in the range.");
   else
      if (guess > 99)
         System.out.println("That's not even in the range.");
      else
         System.out.println("I'm sorry, " + guess + " is wrong.");
}
```

If / else

Here's how the ifs and elses match up.

Keep up this indentation and the program will be unreadable.

```
if (guess == 16)
   System.out.println("That is correct!");
else
   if (guess < 1)
      System.out.println("That's not even in the range.");
   else
      if (guess > 99)
         System.out.println("That's not even in the range.");
      else
         System.out.println("I'm sorry, " + guess + " is wrong.");
```

If / else

Here's the standard way of writing a multibranch `if-else` statement.

Note that `else if` becomes essentially a single thought.

```
if (guess == 16)
   System.out.println("That is correct!");
else if (guess < 1)
   System.out.println("That's not even in the range.");
else if (guess > 99)
   System.out.println("That's not even in the range.");
else
   System.out.println("I'm sorry, " + guess + " is wrong.");
```

If / else

Be careful. . .

```
int number = 15;

if (number > 1)
   if (number > 20)
      System.out.println("Number too big");
else
   System.out.println("Number too small");
```

If / else

The compiler ignores whitespace and matches `else` with the closest `if`.

```
int number = 15;

if (number > 1)
   if (number > 20)
      System.out.println("Number too big");
else
   System.out.println("Number too small");

...

Number too small
```

If / else

These two conditions have the same result.

Shouldn't there be a way to combine them?

```
if (guess == 16)
   System.out.println("That is correct!");
else if (guess < 1)
   System.out.println("That's not even in the range.");
else if (guess > 99)
   System.out.println("That's not even in the range.");
else
   System.out.println("I'm sorry, " + guess + " is wrong.");
```

Combining conditions

Conceptually, we want to combine two conditions.

If the number is less than one or greater than 100...

We would like to produce another boolean value from the ones we have.

Boolean operators

Use boolean operators.

```
int guess = 53;
boolean belowFloor = guess < 1;
boolean aboveFloor = guess >= 1;
boolean belowCeil = guess <= 99;
boolean aboveCeil = guess > 99;
boolean inRange = aboveFloor && aboveCeil;        // and
boolean outOfRange = belowFloor || aboveCeil;     // or

System.out.println(belowFloor + " / " + aboveFloor + " / " +
                   belowCeil + " / " + aboveCeil + " / " +
                   inRange + " / " + outOfRange);
```

```
...

false / true / true / false / false / false
```

Boolean operators

The three boolean operators are and (&&), or (||), and not (!). They can be defined by truth tables.

x	y	x && y
true	true	true
true	false	false
false	true	false
false	false	false

x	y	x || y
true	true	true
true	false	true
false	true	true
false	false	false

x	!x
true	false
false	true

Boolean operators

Our old program, now refined.

```
if (guess == 16)
   System.out.println("That is correct!");
else if (guess < 1 || guess > 99)
   System.out.println("That's not even in the range.");
else
   System.out.println("I'm sorry, " + guess + " is wrong.");
```

Boolean operators

Highest precedence	++, --, unary -, type casting, and !
	*, /, and %
	+ and -
	<, >, <=, and <=
	== and !=
	&&
	||
Lowest precedence	= and friends

Boolean operators

Can you predict these values?

```
boolean x = true,
        y = false,
        z = true;

System.out.println(! x && y);
System.out.println(! x || y);
System.out.println(! x || z);
System.out.println(x && !y);
System.out.println(x && y || x && !z);
System.out.println(x || y && x || !z);
System.out.println((x || y) && !(x && y));
```

Boolean operators

Can you predict these values?

```
boolean x = true,
        y = false,
        z = true;

System.out.println(! x && y);                    false
System.out.println(! x || y);
System.out.println(! x || z);
System.out.println(x && !y);
System.out.println(x && y || x && !z);
System.out.println(x || y && x || !z);
System.out.println((x || y) && !(x && y));
```

Boolean operators

Can you predict these values?

```
boolean x = true,
        y = false,
        z = true;

System.out.println(! x && y);              false
System.out.println(! x || y);              false
System.out.println(! x || z);
System.out.println(x && !y);
System.out.println(x && y || x && !z);
System.out.println(x || y && x || !z);
System.out.println((x || y) && !(x && y));
```

Boolean operators

Can you predict these values?

```
boolean x = true,
        y = false,
        z = true;

System.out.println(! x && y);                 false
System.out.println(! x || y);                 false
System.out.println(! x || z);                 true
System.out.println(x && !y);
System.out.println(x && y || x && !z);
System.out.println(x || y && x || !z);
System.out.println((x || y) && !(x && y));
```

Boolean operators

Can you predict these values?

```
boolean x = true,
        y = false,
        z = true;

System.out.println(! x && y);                  false
System.out.println(! x || y);                  false
System.out.println(! x || z);                  true
System.out.println(x && !y);                   true
System.out.println(x && y || x && !z);
System.out.println(x || y && x || !z);
System.out.println((x || y) && !(x && y));
```

Boolean operators

Can you predict these values?

```
boolean x = true,
        y = false,
        z = true;

System.out.println(! x && y);                 false
System.out.println(! x || y);                 false
System.out.println(! x || z);                 true
System.out.println(x && !y);                  true
System.out.println(x && y || x && !z);        false
System.out.println(x || y && x || !z);
System.out.println((x || y) && !(x && y));
```

Boolean operators

Can you predict these values?

```
boolean x = true,
        y = false,
        z = true;

System.out.println(! x && y);                 false
System.out.println(! x || y);                 false
System.out.println(! x || z);                 true
System.out.println(x && !y);                  true
System.out.println(x && y || x && !z);        false
System.out.println(x || y && x || !z);        true
System.out.println((x || y) && !(x && y));
```

Boolean operators

Can you predict these values?

```
boolean x = true,
        y = false,
        z = true;

System.out.println(! x && y);              false
System.out.println(! x || y);              false
System.out.println(! x || z);              true
System.out.println(x && !y);               true
System.out.println(x && y || x && !z);     false
System.out.println(x || y && x || !z);     true
System.out.println((x || y) && !(x && y)); true
```

Short-circuit evaluation

If the first operand to `&&` is `false` and the first operand to `||` is `true`, Java will not bother evaluating the second operand.

This is called short-circuit evaluation. You can use it to protect against errors.

```
if (possible != 0 && points / possible > 60)
    System.out.println("Average above passing");
```

Example

```
int grade = DocsIO.readint("Enter your numeric grade-->");

System.out.println("Your letter grade:");
if (grade < 60)
   System.out.println("F");
else if (grade < 70)
   System.out.println("D");
else if (grade < 80)
   System.out.println("C");
else if (grade < 90)
   System.out.println("B");
else if (grade <= 100)
   System.out.println("A");
else
   System.out.println("A+");
```

Example

```
double radius = DocsIO.readdouble("Please enter the radius-->");

System.out.println("Please select from the following:");
System.out.println("\t1. Diameter");
System.out.println("\t2. Circumference");
System.out.println("\t3. Area");

int choice = DocsIO.readint("Your choice-->");

if (choice < 1 || choice > 3)
   System.out.println("That was not a valid option.");
else {
   double result;
   if (choice == 1)
        result = radius * 2;
   else if (choice == 2)
        result = (radius * 2) * 3.14159;
   else
        result = radius * radius * 3.14159;
   System.out.println("Result: " + result);
}
```

Comparing Strings

Comparing `Strings` is tricky.

```
String x = "aloha";
String y = x;
String z = "alo";

z += "ha";
System.out.println(x + " " + y + " " + (x==y));
System.out.println(x + " " + z + " " + (x==z));
```

Comparing Strings

What's going on?

```
String x = "aloha";
String y = x;
String z = "alo";

z += "ha";
System.out.println(x + " " + y + " " + (x==y));
System.out.println(x + " " + z + " " + (x==z));
```

```
...

aloha aloha true
aloha aloha false
```

Comparing Strings

On `Strings`, `==` reports whether its operands are the same object (stored in the same part of memory), not whether they contain the same sequence of characters.

It returns `true` if its operands are the exact same `String`, not if they are two `Strings` that happen to be the same.

Comparing Strings

Use the method *string*.equals(*string*).

```
String x = "aloha";
String y = x;
String z = "alo";

z += "ha";
System.out.println(x + " " + y + " " + (x==y));
System.out.println(x + " " + z + " " + (x==z));
System.out.println(x + " " + z + " " + x.equals(z));
```

```
...

aloha aloha true
aloha aloha false
aloha aloha true
```

Warnings

- Don't mix up = and ==.
- Watch out for missing {.
- Declare variables in the right place.
- Be careful about initializing variables in branches.
- Make sure your `ifs` and `elses` match as you expect.
- Remember `&&` has higher precedence than || (or just use parentheses).
- Use `.equals()` to compare `Strings`.

Summary

Be able to identify the following concepts:

- `boolean` type
- Lexicographical order
- Branch statement
- Flow of control
- Block statement
- Scope
- Multibranch
- Short-circuit evaluation