Lesson 39: Conditionals #3 (W11D4)

Balboa High School

Michael Ferraro

October 29, 2015

Do Now

In order to qualify for a $50k loan, the following conditions must be met:

- Your annual income must be at least $26,000
- Your age must be between 18 and 29 (inclusive)

Write a `boolean` that would serve as an appropriate condition for the `if()` statement below.

```
int income, age; //assume these values are set...

Loan myLoan = new Loan(50000);

if( _________________ ) {
    myLoan.grant();
} else {
    myLoan.deny();
}
```

Aim

Students will learn about *Short-Circuit Evaluation*, `if()/else if`, and work on programming exercises.

Short-Circuit Evaluation

Consider this scenario:

```
boolean a = false, b = true, c = true;

if ( a && ( b || c ) ) {
    // statements to run if true...
}
```

How does Java evaluate the condition?

Short-Circuit Evaluation

Consider this scenario:

```
boolean a = false, b = true, c = true;

if ( a && ( b || c ) ) {
    // statements to run if true...
}
```

How does Java evaluate the condition?

→ *In an* `and` *statement, if the left side is false, there's no way the overall truth value could possibly be* `true`*, so why bother to spend CPU cycles to evaluate the rest?*

Short-Circuit Evaluation

Consider this scenario:

```
boolean a = false, b = true, c = true;

if ( a && ( b || c ) ) {
    // statements to run if true...
}
```

How does Java evaluate the condition?

→ *In an* `and` *statement, if the left side is false, there's no way the overall truth value could possibly be* `true`*, so why bother to spend CPU cycles to evaluate the rest?*

This shortcut is called Short-Circuit Evaluation.

Short-Circuit Evaluation for `or`

How might the principle of *Short-Circuit Evaluation* be applied to `or`?

Short-Circuit Evaluation for `or`

How might the principle of *Short-Circuit Evaluation* be applied to `or`?

Ex: You have to be at least 25 or have a military ID to rent a car.

Short-Circuit Evaluation for `or`

How might the principle of *Short-Circuit Evaluation* be applied to `or`?

Ex: You have to be at least 25 or have a military ID to rent a car.

```
int age = 19;
boolean inMilitary = true;

if ( inMilitary || age >= 25 ) {
    ...
}
```

Short-Circuit Evaluation for `or`

How might the principle of *Short-Circuit Evaluation* be applied to `or`?

Ex: You have to be at least 25 or have a military ID to rent a car.

```
int age = 19;
boolean inMilitary = true;

if ( inMilitary || age >= 25 ) {
    ...
}
```

→ *In an* `or` *statement, if the left side is true, then there's no way the entire* `or` *statement won't be true — so don't bother evaluating the rest.*

Short-Circuit Evaluation for Safety!

Let’s say you have this condition to evaluate:

```
a/b > 6
```

Short-Circuit Evaluation for Safety!

Let's say you have this condition to evaluate:

```
a/b > 6
```

What is the potential problem with evaluating this boolean expression? (*Consider the values that the variables might have!*)

Short-Circuit Evaluation for Safety!

Let's say you have this condition to evaluate:

```
a/b > 6
```

What is the potential problem with evaluating this boolean expression? (*Consider the values that the variables might have!*)

→ `b` must not be zero or else there will be a "divide by zero" exception!

Short-Circuit Evaluation for Safety!

Let's say you have this condition to evaluate:

```
a/b > 6
```

What is the potential problem with evaluating this boolean expression? (*Consider the values that the variables might have!*)

→ `b` must not be zero or else there will be a "divide by zero" exception!

Short-circuit evaluation prevents that case:

```
b != 0 && a/b > 6
```

`if()`-wrapped `if()`

- What if you need to branch in more than two directions based on a condition?

[1] \$3 return/payout on \$2 bet

if()-wrapped if()

- What if you need to branch in more than two directions based on a condition?

- Example: The game of Blackjack

  You are dealt two cards. Based on the sum of their values, here are possible actions:

sum	**action**
21 (e.g., A♡ & K♣)	Dealer pays 3:2[1]
10 (e.g., 3♢ & 7♠)	Double down
<10 (e.g., 5♣ & 2♡)	Hit

[1] $3 return/payout on $2 bet

if()-wrapped if()

How one might implement the Blackjack logic from the last slide:[2]

```
int cardOneVal, cardTwoVal;
// call method to deal cards, set card values

int sum = cardOneVal + cardTwoVal;

if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
}
```

[2] cases grossly oversimplified!

if()-wrapped if()

How one might implement the Blackjack logic from the last slide:

```
int cardOneVal, cardTwoVal;
// call method to deal cards, set card values

int sum = cardOneVal + cardTwoVal;

if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
} else {
    //2 more cases:
    // * sum is 10,
    // * sum is less
}
```

if()-wrapped if()

How one might implement the Blackjack logic from the last slide:

```
int cardOneVal, cardTwoVal;
// call method to deal cards, set card values

int sum = cardOneVal + cardTwoVal;

if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
} else {
    if ( sum == 10 ) {
        player.doubleDown();
        dealer.dealCard();
    }
    // * sum is less
}
```

if()-wrapped if()

How one might implement the Blackjack logic from the last slide:

```
int cardOneVal, cardTwoVal;
// call method to deal cards, set card values

int sum = cardOneVal + cardTwoVal;

if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
} else {
    if ( sum == 10 ) {
        player.doubleDown();
        dealer.dealCard();
    } else {
        dealer.dealCard();
    }
}
```

if()-wrapped if()

```
if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
} else {
    if ( sum == 10 ) {
        player.doubleDown();
        dealer.dealCard();
    } else {
        dealer.dealCard();
    }
}
```

- What we have here are *nested* `if()` *statements*

`if()`-wrapped `if()`

```
if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
} else {
    if ( sum == 10 ) {
        player.doubleDown();
        dealer.dealCard();
    } else {
        dealer.dealCard();
    }
}
```

- What we have here are *nested* `if()` *statements*
- Such a construct is often hard to understand and debug

if()-wrapped if()

```
if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
} else {
    if ( sum == 10 ) {
        player.doubleDown();
        dealer.dealCard();
    } else {
        dealer.dealCard();
    }
}
```

- What we have here are *nested* `if()` *statements*
- Such a construct is often hard to understand and debug
- Better way: `else if()`

else if()

Here's the same logic using else if():

```
if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
}
```

else if()

Here's the same logic using `else if()`:

```
if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
} else if ( sum == 10 ) {
    player.doubleDown();
    dealer.dealCard();
}
```

else if()

Here's the same logic using else if():

```
if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
} else if ( sum == 10 ) {
    player.doubleDown();
    dealer.dealCard();
} else { //all other cases
    dealer.dealCard();
}
```

else if()

```
if ( sum == 21 ) {
    dealer.payOut(3,2); //winner!
} else if ( sum == 10 ) {
    player.doubleDown();
    dealer.dealCard();
} else { //all other cases
    dealer.dealCard();
}
```

- Indentation makes the code more readable, easier to follow
- You can have as many `else if()` statements as you need to cover all possible outcomes for which you need different behaviors
- You'll learn `switch/case/break` soon, which is essentially a special form of what you see above

Programming Exercises

- You can find the PDF of Ch. 6 here.
- Either alone or with a partner, write classes that include these methods, and place them in a new project called `Lesson39`.
  - Develop `isLeapYear()` as per **Ch. 6, #13**. Make sure you understand the requirements for a leap year by reading the description carefully!
  - Develop `findBestFit()` as per **Ch. 6, #19**. A tester class is available here.

HW

Finish §3 of PS #6 by next class. Come with questions about book problems you're unable to figure out!