

Unit II: Creating Code

Chapter 12: Boolean Expressions

In this lesson, you will use Boolean Expressions. Boolean expressions are ones that can be evaluated as true or false. Sometimes a programmer would like one statement, or group of statements to execute only if certain conditions are true. There may be a different statement, or group of statements that are to be executed when the condition is false. An expression that can be evaluated to true or false is called a **Boolean expression**. *(Named after a mathematician named Boole who developed Boolean algebra.)* You will use if/else control structures to control which code to execute.

Objectives

- Write code to input answers of various types from the user;
- Use the function **getBooleanFromUser** to get true or false answers;
- Write code to compare values and take appropriate actions;
- Be able to use **AND** and **OR** in Boolean expressions;
- Use the function **getDistanceTo** to find which of two animals is closest to an object.

Boolean Variables

We have a scene in a room with Alice, The Queen of hearts, and a table with a cake and a teapot. In scene view we will make the cake and the teapot invisible. The queen will ask Alice if she is hungry. *(The user will answer for Alice.)* If the user selects **true**, we will make the cake visible. If the user selects **false**, nothing happens. Then we do the same with the teapot.



Code

```
public void myFirstMethod() {
    // ALice and Queen
    // Programmer : Janet Joy
    // The cake and teapot are invisible, opacity = 0.0
    Boolean hungry = this.queenOfHearts.getBooleanFromUser( "Are you hungry?";
    if( hungry ) {
        this.cake.setOpacity( 1.0 );
    } else {
        this.queenOfHearts.say( "Too bad, there's a cake around here somewhere.",
            Say.duration( 2.0 ) );
    }
}
```

Study the code, then try adding the code to ask Alice if she is thirsty, then make the teapot appear if she is thirsty.

If/Else

The format for the if else control structure is as follows:

if(Boolean expression) {

Statements to execute when the Boolean expression is true.

}

else {

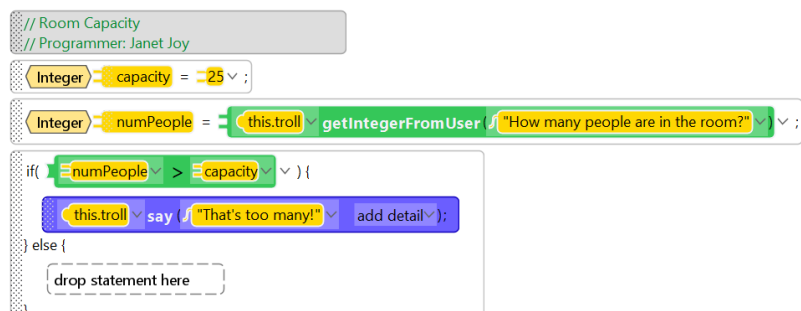
Optional : Statements to execute when the Boolean expression is false.

}

Note that a Boolean variable is a Boolean expression, you do not need to compare it to anything.

Integer Variables

Integer variables or values need to be compared with other integer values to create a Boolean expression. Here we compare the number of people and the room capacity:



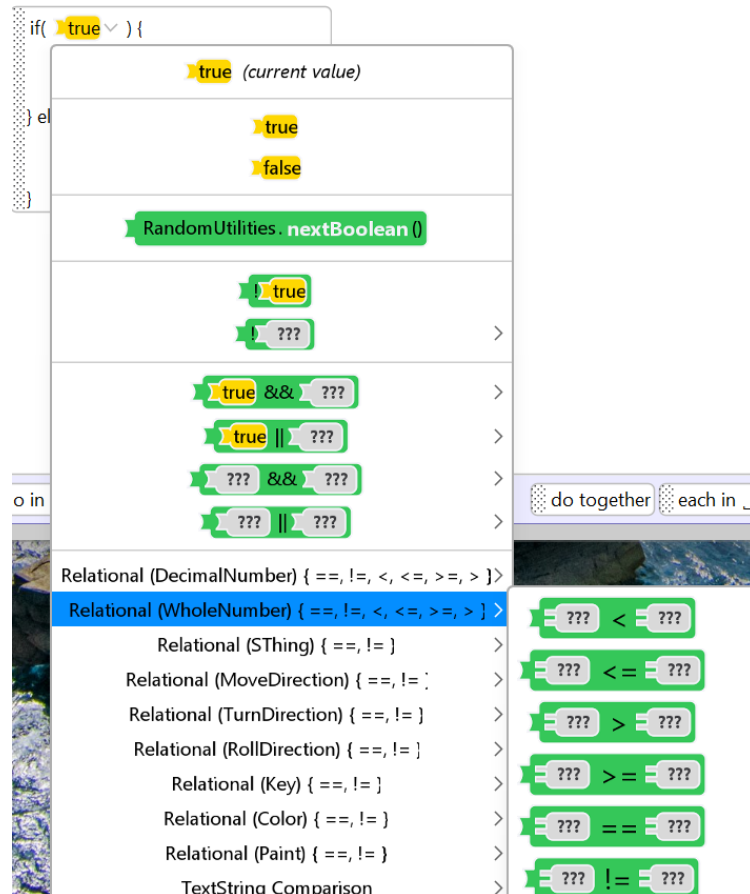
if(**numPeople>capacity**) the troll says, “That’s too many.”

numPeople>capacity is a **Boolean expression**: it is either **true** or **false**.

We have not included an else in this example.

Integer Comparison

To write this statement we first select if(true) then Relational (Whole Number) then we select a comparison, then the two variables.



Symbol	Meaning	True Examples	False Examples
<	Less than	$25 < 30$, $30 < 31$, $9 < 10$	$25 < 25$, $25 < 23$
<=	Less than or equal	$25 <= 30$, $25 <= 25$	$3 <= 2$, $30 <= 31$
>	Greater than	$30 > 25$, $5 > 4$	$30 > 30$, $30 > 31$
>=	Geater than or equal	$30 >= 30$, $30 >= 25$	$30 >= 40$, $0 >= 5$
==	Equal (the same)	$30 == 30$, $-3 == -3$	$30 == 31$, $30 == 2$
!=	Not Equal	$30 != 31$, $30 != 2$	$30 != 30$, $-3 != -3$

Notice that the same symbols can be used to compare Relational (Decimal Numbers) but you cannot compare an integer and a decimal number.

You can convert an integer to a double, or a double to an integer using round, or other functions.

Double Comparison Example

In this example two monkeys, Pete and Penny, see a banana. We declare the **Double** variable **peteDistance** with a placeholder value, then drag the function **getDistanceTo** to replace the placeholder value. We select the banana as the argument to the **getDistanceTo** function. We do the same to declare and give a value to **pennyDistance**.

If **pennyDistance** is greater than or equal to **peteDistance**, Pete tells her she can have the banana. If Pete felt greedy, he could have only given the banana to Penny, if her distance was greater than > his.

Tip: There are several statements in this program that were copied to the clipboard, then dragged in and modified. This is much faster, especially when you have to build a statement piece by piece.

```
void myFirstMethod ( )
do in order
    // Which monkey is closer to the banana
    // Programmer: Janet Joy
    // You can use the clipboard for a lot of this code.

    Double peteDistance = this.pete.getDistanceTo ( this.banana );

    Double pennyDistance = this.penny.getDistanceTo ( this.banana );

    ThreadUtilities.doTogether( ()-> {
        this.pete.say ( "Look! A banana!", Say.duration ( 2.0 ) );
    }, ()-> {
        this.penny.say ( "Look! A banana!", Say.duration ( 2.0 ) );
    } );

    this.penny.say ( "My distance to the banana is " + pennyDistance );
    this.pete.say ( "My distance to the banana is " + peteDistance );

    if ( pennyDistance >= peteDistance ) {
        this.pete.say ( "You're closer, you can have it." );
        this.penny.say ( "Thank you." );
    } else {
        this.penny.say ( "You're closer, you can have it." );
        this.pete.say ( "Thank you." );
    }
}
```

String Comparison

We have a scene with an alien and a UFO, and a woman and a Humvee. Somehow, the alien speaks English.



There are several ways to compare strings. In this first example, location (“In the desert, on planet earth.” does not equal “Earth” and the alien thinks he’s lost.

```
// Alien and Woman have conversation
// Programmer: Janet Joy

String location = "In the desert, on planet earth." ;

this.alien say "Where am I?" add detail ;

this.woman say location add detail ;

if ( location contentEquals "Earth" ) {
    this.alien say "Good, that's where I'm supposed to be." add detail ;
} else {
    this.alien say "Wow, I'm really lost!" add detail ;
}
```

Here, we ask if the location contains the string “earth” and it does, so the alien says he’s where he’s supposed to be. There are several ways to compare strings, as shown below:

```
??? contentEquals ( ??? ) >
??? equalsIgnoreCase ( ??? ) >
??? startsWith ( ??? ) >
??? endsWith ( ??? ) >
??? contains ( ??? ) >
```

Comparison with AND and OR

Logical AND: &&

With **AND**, **both conditions must be true** for the entire expression to be true.

Everyday situation	Conditions
Logging into an account	Username is correct AND password is correct
Getting a driver's license	Person is old enough AND passes the driving test
Going swimming	Pool is open AND you can swim
Receiving a discount	Customer is over 65 AND has a membership card
Passing a course	Student has a passing grade AND completed required assignments
Going on an amusement park ride	Rider is tall enough AND old enough

Logical OR: || (Note: on the keyboard, this symbol is shift+\)

With **OR**, **at least one condition must be true**. If both are true, that is OK.

Everyday situation	Conditions
Get a traffic ticket	Speeding OR aggressive driving (if both they get a ticket)
Getting a discount	Customer is at least 65 OR younger than 12. (They can't be both, but either one gives them a discount.)

Note: In English, we often use the word **or** to mean a choice: Do you want coffee or tea? Are you taking the train or bus? Do you want a hamburger or hot dog? The meaning of this is **one or the other, not both**. In programming this is called an exclusive **OR**, or **XOR**.

Truth Table:

You can only go up in the hot air balloon if you are at least 15 and have a ticket. We will create a truth table to show the possibilities. Instead of trying to enter all possible ages, we will just check the boundary ages. The idea is to test values **just below, exactly at, and just above** an important boundary.

Age	Has Ticket	Age>=15 && hasTicket
14	true	false
14	false	false
15	true	true
15	false	false
16	true	true
16	false	false

Here, the mad hatter oversees an amusement park ride. He will ask the boy his age and if he has a ticket.



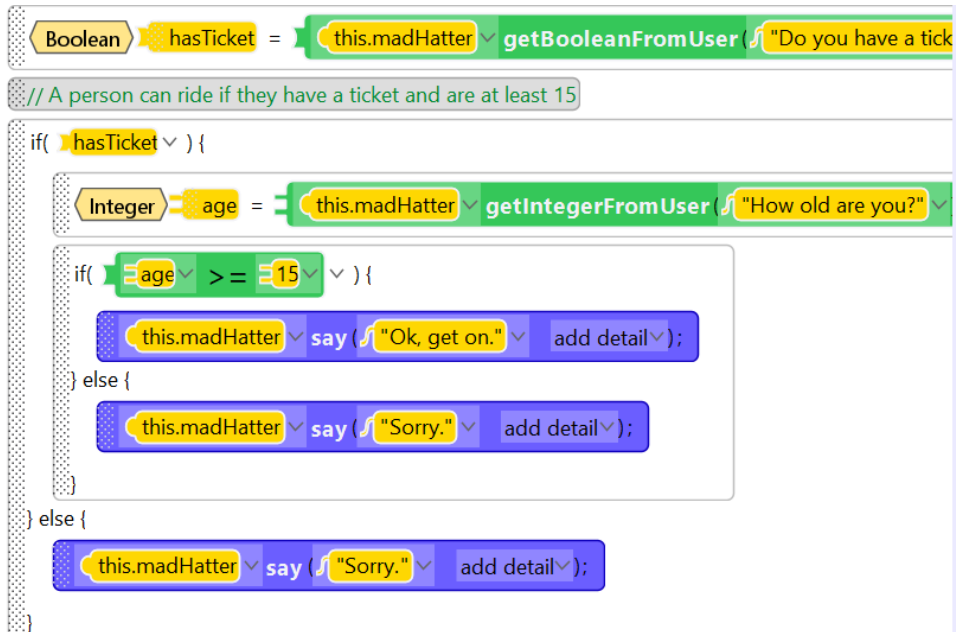
```

Boolean hasTicket = this.madHatter.getBooleanFromUser("Do you have a tick
Integer age = this.madHatter.getIntegerFromUser("How old are you?" );
// A person can ride if they have a ticket and are at least 15
if( hasTicket && age >= 15 ) {
    this.madHatter.say("Ok, get on." add detail);
} else {
    this.madHatter.say("Sorry." add detail);
}

```

Here we ask them if they have a ticket and also ask their age. We can improve this as shown below: First, we ask if they have a ticket. If they have a ticket, we ask them their age.

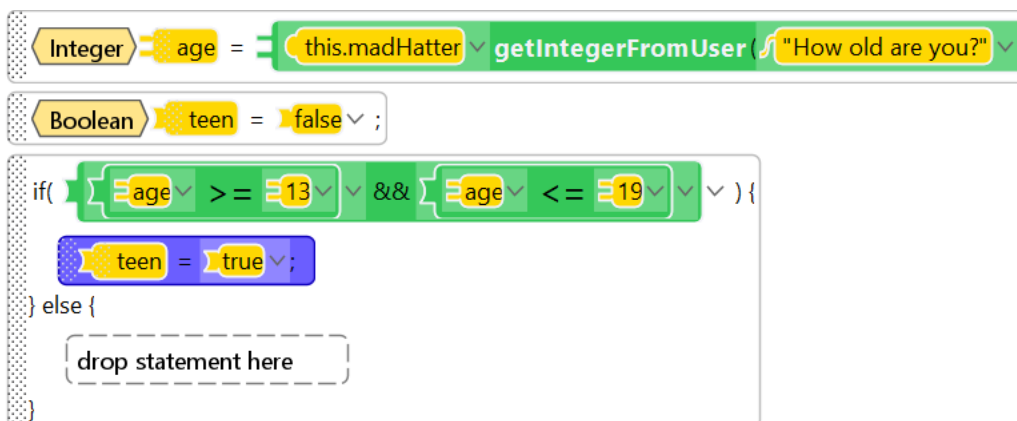
If they don't have a ticket, we don't ask their age, just say "Sorry"



AND: Teenager:

A teenager is between the ages of 13 and 19 inclusive. The symbol for and is &&

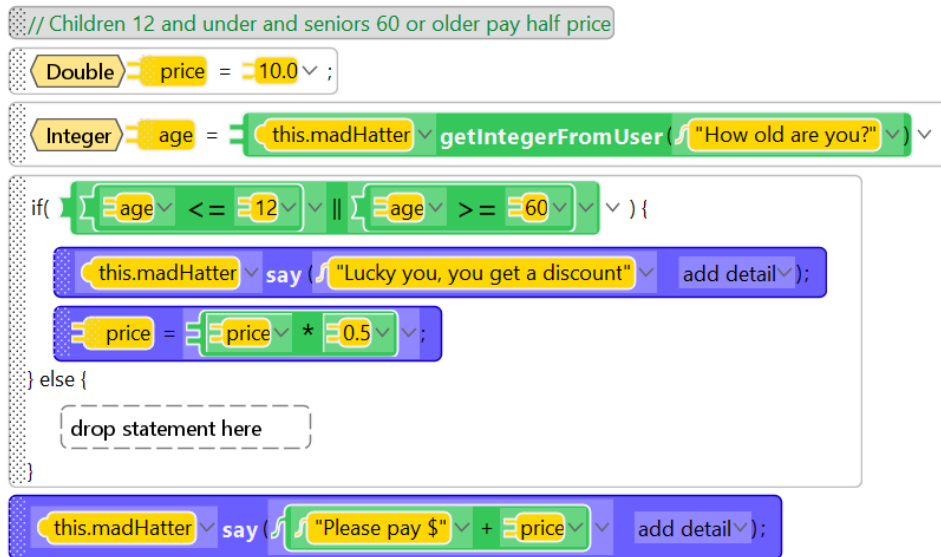
In code we write this as **age>=13 && age<=19**. Notice that both sides of the && must be a complete Boolean expression.



We don't need an **else** statement here, because we initialized **teen** to **false** and only set it to true if the age is in the right range. However, it might make the code clearer to add the else statement **teen=false;**

OR: Children and Seniors get a discount:

We may say that children and seniors get a discount, but one individual can only fit one of those categories. If we **write if(age<=12 && age>=60)** no one will get a discount because no one person is less than 12 and older than 60. The correct code is shown below:



Truth Table:

The symbol for **or** is `||`. This truth table compares OR: `||` and AND: `&&` for all possible values for expressions A and B. Boolean expressions can only have values of true or false, no matter what values we are comparing.

Expression A	Expression B	A B	A && B
true	true	true	true
True	false	true	false
false	true	true	false
false	false	false	false

Here is the truth table showing the age values and who gets a discount. We test each of the boundary values.

Age	Age<=12	Age>=60	Gets Discount
11	true	false	true (child)
12	true	false	true (child)
13	false	false	false
59	false	false	false
60	false	true	true (senior)
61	false	true	true (senior)