

## September 25, 2013: Announcements

Assignment 2 is due tonight at 9:00 pm. Please don't wait until the last few hours to submit and test your code. WebGS cannot handle 400 students submitting at one time.

Assignment 3 is due next Wednesday (Oct 2<sup>nd</sup>) at 9:00 pm.

Keep using Piazza! One update though. It will be more efficient if you email your Section Leader with questions about your code in particular.

Office Hours: 1-3pm on Monday and Wednesday (4 total hours of office hours)

Assignment 4 has been posted.

### **Flow of Control:**

Programs can control the order in which their instructions are executed.

A major feature of software and why it is useful!

Four types of flow:

Sequential:

Execute instructions in the order listed in the code. (This is mostly what you have been doing.)

Method calls:

Transfer flow control to the code inside the method; i.e.

```
String zap = "Once upon a time in a galaxy far, far  
away...";  
String word = zap.substring(22, 28);  
System.out.println("word contains: " + word);
```

Control returns back to the point of the call. Some calls also return a value.

### **Flow of Control:**

Programs can control the order in which their instructions are executed.

A major feature of software and why it is useful!

Four types of flow:

Selection: (Chapter 5 — now!)

Which set of instructions are executed depends on the data.

Looping: (Chapter 6)

Repeat a set of instructions, changing some of the data each time through the set of instructions.

### **Flow of Control:**

When would we use flow of control? What would that look like?

Write a program that would decide which line people should use at the grocery store based on how many items they have in their cart.

Ask the user for how many items they have

Evaluate the input: print a message that is appropriate for the data entered

If they have 15 items or less, tell the user to go to the checkout line

If they have more than 15 items, tell the user to go in the normal line

What are some other scenarios where we might want to do something like this?

My example had only two options, so that it could be short. That is not a requirement of Flow of Control. Later on we'll see examples that are much more complex.

### Comparison Operators:

Reading: Section 5.1

In the previous example, I said things like, “If the user has 15 items or less...” What does that look like in code?

We need ways to compare values. So, I need to be able to take the temperature that the user gave me and look at it and decide what range it’s in, so that I can tell the user what kind of clothes are appropriate.

We will then use these techniques to make choices about which instructions to execute.

All of these techniques have one goal: to compute a **boolean** value (**true** or **false**).

### Comparison Operators:

Equality, or the lack thereof:

There are two “equality” operators:

**==** compares two values and returns **true** if one is equal to the other.

**!=** compares two values and returns **true** if one is not equal to the other.

Example:

```
boolean answer1, answer2;
Scanner inputScan = new Scanner( System.in );
int aNum, bNum;
System.out.print("Enter two integers: ");
aNum = inputScan.nextInt();
bNum = inputScan.nextInt();
answer1 = (aNum == bNum);
answer2 = (aNum != bNum);
```

Comparison Operators (continued):

Equality Operators (continued):

Do not confuse these two:

**==** for comparing two values.

**=** for assigning a value to a variable.

Examples:

```
int aNum = 42, bNum = 96;
```

```
boolean answer;
```

```
answer = (aNum = bNum);
```

Gives the error: "Incompatible types". Why?

```
int zap;
```

```
zap = (aNum = bNum);
```

```
System.out.println("zap is " + zap);
```

Compiles without error.  
Runs without error.  
What is printed?

Comparison Operators (continued):

Relational Operators:

Result is boolean: **true** or **false**

Relational Operators	Type	Meaning
<	binary two operands	is less than
<=	binary two operands	is less than or equal to
>	binary two operands	is greater than
>=	binary two operands	is greater than or equal to

Comparison Operators (continued):

**Relational Operators** (continued):

Example:

```
import java.util.Scanner;

public class Relational
{
    public static void main(String[] args)
    {
        boolean answer;
        Scanner inputScan = new Scanner( System.in );
        int aNum, bNum;

        System.out.print("Enter two integers: ");
        aNum = inputScan.nextInt();
        bNum = inputScan.nextInt();
```

Comparison Operators (continued):

**Relational Operators** (continued):

Example (continued):

```
        System.out.println("Result of (aNum < bNum) is " +
                           (aNum < bNum));
        System.out.println("Result of (aNum <= bNum) is " +
                           (aNum <= bNum));
        System.out.println("Result of (aNum > bNum) is " +
                           (aNum > bNum));
        System.out.println("Result of (aNum >= bNum) is " +
                           (aNum >= bNum));

    } // end of method main
} // end of class Relational
```

Comparison Operators (continued):

**Relational and Equality Operators** (continued):

The two-character operators (**==**, **!=**, **<=**, **>=**):

No space in-between the two characters.

The order of the two characters does matter (except for **==**, of course :-):

```
boolean result = (aNum => bNum) ;
```

Will not compile, gives the error: “illegal start of expression”

Comparison Operators (continued):

Let’s go back to this example:

Write a program that would decide which line people should use at the grocery store based on how many items they have in their cart.

Ask the user for how many items they have

Evaluate the input: print a message that is appropriate for the data entered

If they have 15 items or less, tell the user to go to the checkout line

If they have more than 15 items, tell the user to go in the normal line

Now that we’ve seen conditional operators, we can think about what code we would have to write in order to make this happen

### Comparison Operators (continued):

Let's take the steps and translate them to code:

Ask the user for how many items they have

```
Scanner keyboard = new Scanner(System.in);  
System.out.print("Enter how many items you have: ");  
int numItems = keyboard.nextInt();
```

Evaluate the input: print a message that is appropriate for the data entered

If they have 15 items or less, tell the user to go to the express checkout line

```
boolean useExpress = numItems <= 15;
```

If they have more than 15 items, tell the user to go in the normal checkout line

```
boolean useNormal = numItems > 15;
```

Now that we know how to express our conditions, how do we check them?

### Simple if Statements:

Reading: Section 5.2

Used when the program should perform an operation for one set of data, but not for all other data.

This is how we could do something only when a customer has 15 items or less, for example

Syntax:

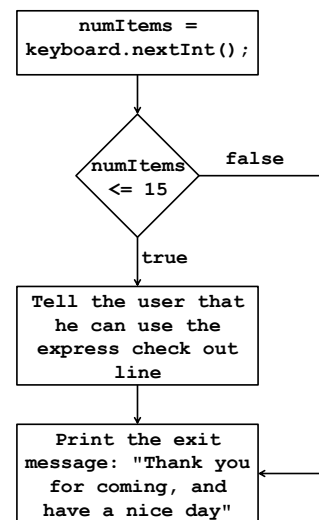
```
if ( condition-goes-here ) {  
    // true block  
    // code to execute when condition is true  
}  
// code here that executes after the if statement
```

Simple if Statements (continued):

```
Scanner keyboard = new Scanner(System.in);
System.out.print("Enter the number of items: ");
int numItems = keyboard.nextInt();
if (numItems <= 15) {
    System.out.println("You may use the express " +
        "check out line!");
}
// we'll think about the other option in a bit...
System.out.println("Thank you, and have a nice day!");
```

Simple if Statements (continued):

```
Scanner keyboard = new
Scanner(System.in);
System.out.print("Enter the number " +
    " of items: ");
int numItems = keyboard.nextInt();
if (numItems <= 15) {
    System.out.println("You may use " +
        "the express check out line!");
}
System.out.println("Thank you, and " +
    "have a nice day!");
```



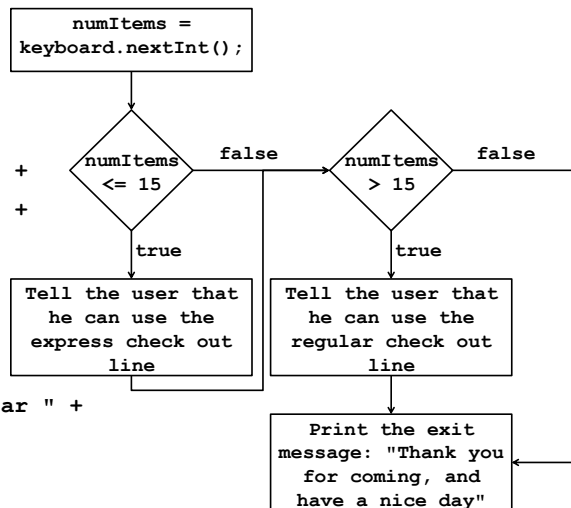


Simple if Statements (continued):

```
Scanner keyboard = new Scanner(System.in);
System.out.print("Enter the number of items: ");
int numItems = keyboard.nextInt();
if (numItems <= 15) {
    System.out.println("You may use the express " +
        "check out line!");
}
if (numItems > 15) {
    System.out.println("You need to use the regular " +
        "check out line.");
}
System.out.println("Thank you, and have a nice day!");
```

Simple if Statements (continued):

```
// to save room I removed
// the lines for Scanner
if (numItems <= 15) {
    System.out.println("You " +
        "may use the express " +
        "check out line!");
}
if (numItems > 15) {
    System.out.println("You " +
        "need to use the regular " +
        "check out line.");
}
System.out.println("Thank you, and " +
    " have a nice day!");
```



Simple **if** Statements (continued):

Style considerations:

The **{** can go

On the next line, underneath the **i** of **if** (The textbook uses this style).

On the same line (This is another common style).

The **}** goes on a line by itself, underneath the **i** of **if**.

Indent the lines inside the **if** statement. Use either a tab, or at least 3 blank spaces.

```
int numItems = keyboard.nextInt();  
if (numItems <= 15) {  
    System.out.println("You may use " +  
                        "the express " +  
                        "check out line!");  
}
```

Simple **if** Statements (continued):

Be careful: Do not put a semi-colon after the condition.

What happens if the code is?

```
int numItems = keyboard.nextInt();  
if (numItems <= 15); ← The mistake  
{  
    System.out.println("You may use the express " +  
                        "check out line!");  
}
```

Simple **if** Statements (continued):

When there is only one statement inside the **if**, the **{ }**'s can be left off.

Example:

```
int numItems = keyboard.nextInt();  
if (numItems <= 15)  
    System.out.println("You may use the express " +  
                        "check out line!");
```

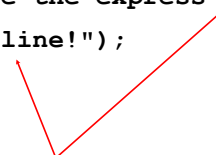
Simple **if** Statements (continued):

Indentation of the code inside an **if** statement is very important for us humans.

But, the Java compiler ignores the indentation.

What happens here?

```
int numItems = keyboard.nextInt();  
if (numItems <= 15)  
    System.out.println("You may use the express ");  
    System.out.println("check out line!");
```



When does this line get executed?

### Comparison Operators (continued):

Let's take look at this example, again:

Ask the user for how many items they have

```
Scanner keyboard = new Scanner(System.in);  
System.out.print("Enter how many items you have: ");  
int numItems = keyboard.nextInt();
```

Evaluate the input: print a message that is appropriate for the data entered

If they have 15 items or less, tell the user to go to the express checkout line

```
boolean useExpress = numItems <= 15;
```

If they have more than 15 items, tell the user to go in the normal checkout line

```
boolean useRegular = numItems > 15;
```

It looks like the boolean conditions for useExpress and useNormal are sort of opposites of each other. Is there a way to take that into consideration?

### Comparison Operators (continued):

#### Logical Operators:

The **!** operator:

Performs a NOT operation.

Has only one operand.

Returns **false** if the operand is **true**. Returns **true** if the operand is **false**.

Example:

```
int numItems;  
numItems = keyboard.nextInt();  
  
boolean useExpress, useRegular;  
  
useExpress = (numItems <= 15);  
  
useRegular = !(numItems <= 15);  
  
useRegular = !useExpress;
```

Both of these give  
the same result.

Simple if Statements (continued):

```
Scanner keyboard = new Scanner(System.in);
System.out.print("Enter the number of items: ");
int numItems = keyboard.nextInt();
if (numItems <= 15) {
    System.out.println("You may use the express " +
        "check out line!");
}
if (!(numItems <= 15)) {
    System.out.println("You need to use the regular " +
        "check out line.");
}
System.out.println("Thank you, and have a nice day!");
```

Simple if Statements (continued):

```
Scanner keyboard = new Scanner(System.in);
System.out.print("Enter the number of items: ");
int numItems = keyboard.nextInt();
boolean useExpress = numItems <= 15;
if (useExpress) {
    System.out.println("You may use the express " +
        "check out line!");
}
if (!useExpress) {
    System.out.println("You need to use the regular " +
        "check out line.");
}
System.out.println("Thank you, and have a nice day!");
```

What's the advantage of doing this?

### Comparison Operators (continued):

Let's take look at this example, yet again:

Ask the user for how many items they have

Evaluate the input: print a message that is appropriate for the data entered

If they have 15 items or less, tell the user to go to the express checkout line

```
boolean useExpress = numItems <= 15;
```

If they have more than 15 items, tell the user to go in the normal checkout line

```
boolean useRegular = numItems > 15;
```

We've seen that we can use the **!** operator to negate a **boolean** condition. But we should strive to write clean and elegant code, not just correct code. So we should see if there is already a construct to let us do exactly this, but in a more elegant way...

### if/else Statements :

Reading: Section 5.3

Allows separate actions for both **true** and **false** results.

The **else** clause contains the code for **false**.

The old model:

One if statement for when the user had 15 items or less

One if statement for when the user had more than 15 items

The new model:

One if/else statement where there are two clauses

A true clause for when the user has 15 items or less

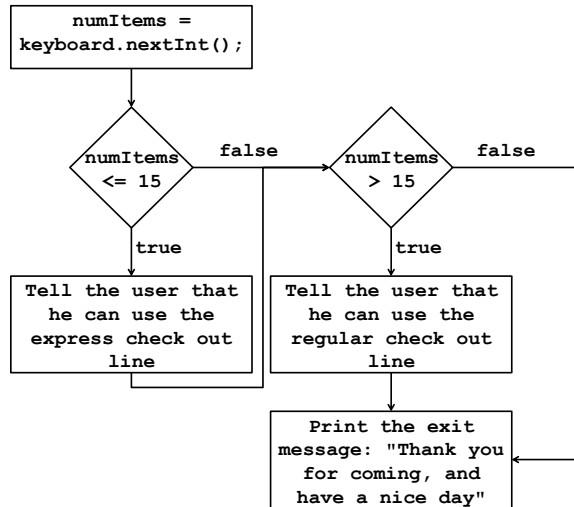
A false clause for when the user has more than 15 items.

### if/else Statements (continued):

The old model:

One if statement for when the user had 15 items or less

One if statement for when the user had more than 15 items



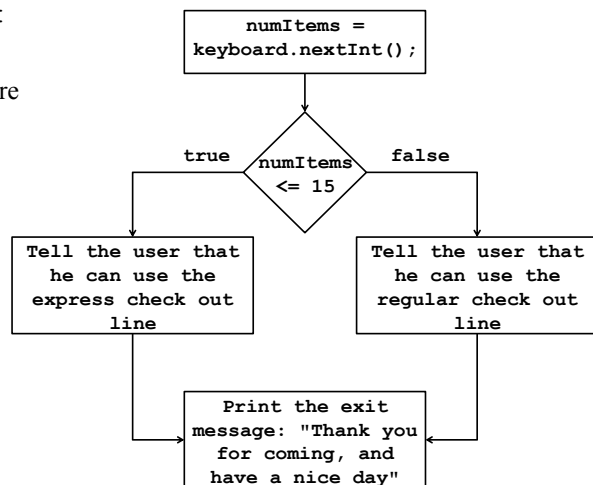
### if/else Statements (continued):

The new model:

One if/else statement where there are two clauses

A true clause for when the user has 15 items or less

A false clause for when the user has more than 15 items.



**if/else Statements** (continued):

Indent the statement(s) inside the **else** clause.

Use a tab, or at least 3 spaces.

When more than one statement is in the **else** clause, you must use { }'s.

Example:

```
if ( numItems <= 15 ) {  
    System.out.println("You may use the express lane!");  
}  
else {  
    System.out.println("You need to use the " +  
        "regular lane.");  
}  
  
System.out.println("Thank you, and have a nice day!");
```

**if/else Statements** (continued):

Indent the statement(s) inside the **else** clause.

Use a tab, or at least 3 spaces.

When more than one statement is in the **else** clause, you must use { }'s.

Example:

```
if ( numItems <= 15 )  
{  
    System.out.println("You may use the express lane!");  
}  
else  
{  
    System.out.println("You need to use the " +  
        "regular lane.");  
}  
  
System.out.println("Thank you, and have a nice day!");
```



Comparison Operators (continued):

Logical Operators (continued):

The **&&** operator:

Performs an AND operation.

Has two operands.

Returns **true** if both operands are **true**; otherwise, **false** is returned.

Example:

```
int age;
age = inputScan.nextInt();

boolean teenAge;

teenAge = (age >= 13) && (age <= 19);

System.out.println("teenAge is " + teenAge);

teenAge = (age > 12) && (age < 20);
```

Comparison Operators (continued):

Logical Operators (continued):

The **&&** operator:

Performs an AND operation.

Has two operands.

Returns **true** if both operands are **true**; otherwise, **false** is returned.

Example:

```
int age;
age = inputScan.nextInt();

boolean teenAge, oldEnough, youngEnough;

oldEnough = age >= 13;
youngEnough = age <= 19;

teenAge = oldEnough && youngEnough;
```

Comparison Operators (continued):

**Logical Operators** (continued):

The `||` operator:

Performs an OR operation.

Has two operands.

There are two ways to think about what the OR operator does:

Returns **true** if either or both operands are **true**; otherwise, **false** is returned.

Returns **false** if both operands are **false**; otherwise, **true** is returned.

Comparison Operators (continued):

**Logical Operators** (continued):

The `||` operator:

Example:

```
int myAge;  
myAge = inputScan.nextInt();  
  
int sisterAge = 34, brotherAge = 39;  
  
boolean notYoungest;  
  
notYoungest = (myAge > brotherAge) || (myAge >  
sisterAge);  
  
System.out.println("notYoungest is " + notYoungest);
```

Comparison Operators (continued):

**Logical Operators** (continued):

Summary in the form of a *truth table*:

a	b	!a	a && b	a    b
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false