CSCI 161
Introduction to Computer Science

Department of Mathematics
and Computer Science

Lecture 3
Writing Classes
Condition Statements

Last Time...

- Self check. You should know:
  - How to *declare* instance variables and local variables
  - How to *assign* values/expressions to variables
  - How to *write* methods and constructors
  - How to *call* existing methods from within a method you're writing

- Still not sure about:
  - *When* to use local variables instead of instance variables
  - *When* and *why* to `return` a value from a method

Operators for Integers and Doubles

- The operators below can be applied to any int or double.
  - Important: These do not change the values of any variables!

- In examples below, assume we start with: `int x = 10;`

Operator	Meaning	Examples	Result
`a + b`	Add **b** to **a**	`x + 3`	`13`
`a - b`	Subtract **b** from **a**	`x - 5`	`5`
`a * b`	Multiply **a** by **b**.	`x * 2`	`20`
`a / b`	Divide **a** by **b** and return the quotient.	`x / 3`	`3`
`a % b`	Divide **a** by **b** and return the remainder! (Applies to int)	`x % 4`	`2`

More: Accumulator Operators

- These operations below *change* the values of the variable on the left.

- In examples below, assume we start with: `int x = 5;`

Operator	Meaning	Examples	Result
`a += b`	Add **b** to the value of **a**.	`x += (6 * 10);`	`65`
`a -= b`	Subtract **b** from **a**.	`x -= 5;`	`0`
`a *= b`	Multiply **a** by a factor of **b**.	`x *= 2;`	`10`
`a /= b`	Divide **a** by **b**.	`x /= 3;`	`1`
`a++`	Increment **a** by 1.	`x++;`	`6`
`a--`	Decrement **a** by 1.	`x--;`	`4`

Ticket Machines as Software

- Ticket machines can be found in most subway and train stations

- What all TicketMachines to:
  - Have a set price/cost for a ticket
  - Print a ticket after user inserts correct money
  - Keep a running total of money collected
  - Assume: Machines only have one ticket price and the price is in whole dollars (too lazy to count cents)

TicketMachine Demo

- Let's first take a look at a demonstration of how we expect the TicketMachine to behave.

- The final code package is provided to you on Canvas.

- [We'll start by writing the class on the board today]

Outline

- Writing Our First Class: TicketMachine
  - Instance Variables
  - Constructors
    - Parameters
  - Methods
    - Printing to screen
    - Local Variables
    - If-Statements
  - Using TicketMachines in Code (no more point and click!)
  - More practice with if-then-else
    - Logical operators
- Conclusion

Ticket Machine: Instance Variables

- We'll name the class **`TicketMachine`**

```
public class TicketMachine
{
   // Declare instance variables here
   // Write constructors here
   // Write methods here
}
```

- What *instance variables* should *all* ticket machines have?
  - Amount of money inserted so far (we'll call that the ***`balance`***)
  - Amount of money accumulated over time by the machine (called ***`total`***)
  - Price per ticket (called ***`price`***)

Outline

- Writing Our First Class: TicketMachine
  - Instance Variables
  - Constructors
  - Methods
    - Printing to screen
    - Local Variables
    - If-Statements
  - Using TicketMachines in Code (no more point and click!)
  - More practice with if-then-else
    - Logical operators
- Conclusion

TicketMachine Constructors

- Recall the general syntax to write a constructor

```
public ClassName(list-of-parameters) {
   //code to initialize instance variables
}
```

- Write two constructors for TicketMachine:
  - One that lets the user set the price for each ticket.
  - And a "default" (no input) constructor that sets the price of a ticket to a random number between $1 and $5.

Random Number Generation

- Before we can generate a random number, we need to import some code at the top of your file!

```
import java.util.Random;
```

- Then, in the body of your code, create a local variable that can store a Random number generator object.

```
Random rng;            // Declare a local variable of data-type Random
rng = new Random();    // Creates a Random object and assigns it to rng
int x = rng.nextInt(1,6);  // Ask rng to run nextInt() method with the given bound
```

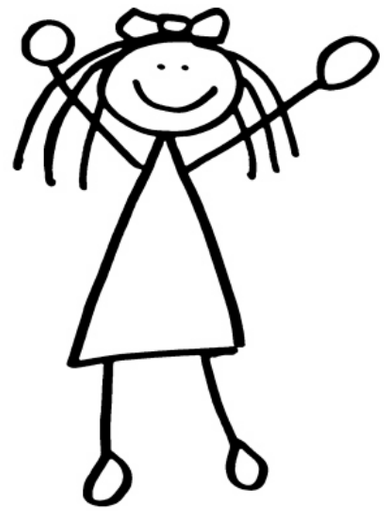
Outline

- Writing Our First Class: TicketMachine
  - Instance Variables
  - Constructors
    - Parameters
  - Methods
    - Printing to screen
    - Local Variables
    - If-Statements
  - Using TicketMachines in Code (no more point and click!)
  - More practice with if-then-else
    - Logical operators
- Conclusion

Ticket Machine Methods

- What *actions* should all ticket machines have?
  - Let a user insertMoney. It should also return the current balance.
  - Retrieve the cost of a ticket: getPrice
  - Retrieve the current balance: getBalance
  - Let a user printTicket. This should also update total and clear the balance.

What can a ticket machine *do*?



Getters and Setters

- Some methods are only meant to set & get instance variables' values
  - *Getters (or Accessors)*
    - What we call methods that simply return (get) a instance variable's value
  - *Setters (or Mutators)*
    - What we call methods that simply change (set) an instance variable's value

- Go ahead and write these getters and setters
  - `getPrice()`
  - `getBalance()`
  - `insertMoney(int amount)`

What Should `insertMoney()` Do?

- Does `insertMoney()` accept inputs?
  - Yes, it should input an integer that we'll call amount
- Does `insertMoney()` return a value to the caller?
  - Yes, it should return the balance after the amount has been inserted

- With the information above, we can write the method signature:

```
public int insertMoney(int amount) {



}
```

What Should `insertMoney()` Do? (2)

- Does `insertMoney()` accept inputs?
  - Yes, it should input an integer that we'll call amount
- Does `insertMoney()` return a value to the caller?
  - Yes, it should return the balance after the amount has been inserted

- Final:

```
public int insertMoney(int amount) {

    balance += amount;  // balance needs to accumulate the given amount
    return balance;
}
```

Outline

- Writing Our First Class: TicketMachine
  - Fields
  - Constructors
  - Methods
    - Returning
    - Printing to screen
    - Local Variables
    - If-Statements
  - Using TicketMachines in Code (no more point and click!)
  - More practice with if-then-else
    - Logical operators
- Conclusion

A New Method: Refund

- People have been requesting that our `TicketMachine` handle refunds!
- Think about what it needs to support:
  - We name the method `refundBalance()`
  - It **returns** the current balance
  - It resets the current balance to zero

- What's ***wrong*** with the following code?

```
public int refundBalance() {
    return balance; //return current balance to user
    balance = 0;    //clear the balance
}
```

Recall that Returning Immediately Exits!

- Returning causes a method to exit!! So does this work?

```
public int refundBalance() {
    balance = 0;    //clear the balance
    return balance; //return current balance to user
}
```

Recall that Returning Immediately Exits!

- No, 0 would always be returned!

```
public int refundBalance() {
    balance = 0;    //clear the balance
    return balance; //return current balance to user
}
```

- **Solution:** We need a local variable to hold the `balance` before resetting it to zero!

```
public int refundBalance() {
    int refund = balance;   // save it first!
    balance = 0;            // ok, now clear the balance
    return refund;
}
```

What Should `printTicket()` Do?

- **Step 1:** We want it to print the following to the screen:

```
##########################
# The Puget Sound Line
# Ticket
# 5 dollars.
##########################
```

This number must reflect the cost of a single ticket at the particular machine
(Hey, we have a field remembering that value)

- **Step 2:** After printing, it should clear update the total and clear the balance.

How to Print Something to the Screen?

- Syntax: `System.out.println(thing-you-want-printed);`

  By the way, `thing-you-want-printed` could also be a variable that's storing a String

- Examples:

```
System.out.println("Hello World!");
> Hello World    <------ This is what appears on the terminal!
```

```
String str = "Hello World!";
System.out.println(str);
> Hello World    <------ This is what appears on the terminal!
```

```
String str = "Hello\n\tWorld!";    // Note: "\n" means new line, "\t" means tab
System.out.println(str);

> Hello
>     World
```

Important: Concatenating Strings

- To *concatenate* is a fancy way of saying, "To append"
  - We can append a String to other Strings, an expression, variables, etc.
  - The concatenation operator is the "plus" symbol: +

- Example:

```
int x = 100;
System.out.println("The value of x is " + x);

> The value of x is 100
```

- Another:

```
int x = 13 * 4 + 9;
System.out.println("13 * 4 + 9 is\n:" + x);

> 13 * 4 + 9 is
> 61
```

Important: Concatenating Strings! (Cont.)

- The *concatenation assignment* symbol **+=** can also be used to build up a String.

- Example: Build a String variable, then print it out!

```
String str = "University";
str += " ";
str += "of";
str += " ";
str += "Puget Sound";
System.out.println(str);

> University of Puget Sound
```

But printTicket() is Broken!

- Current code:

```
public void printTicket() {
    System.out.println("########################");
    System.out.println("# The Puget Sound Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " dollars.");
    System.out.println("########################");

    total += price;
    balance -= price;
}
```

- `printTicket()` lets you to print a ticket no matter how much $ you've put in.
- Instead, we need it to *make a choice*:
  - Is there enough in the **balance** to purchase a ticket?
  - If so, print a ticket. If not, print the amount that still needs entered.

Conditional Statements

- What if we need to make a decision (branch) in our code?

- This is known as an If-Then-Else clause:
  - The **`else`** clause is *optional*, but it is needed in our case.
  - **Java's if-then-else syntax:**

```
if (some-boolean-condition) {
    // statements to execute if
    // the condition was true
}
else {
    // statements to execute if
    // the condition was false
}
```

What is a boolean condition?

What are Boolean Conditions?

- Commonly, they are comparisons that result in a true or false value.
  - Comparison operators
    - Below, a and b can be variables or expressions that evaluate to a number

Comparison Operator	Meaning	Caution
`if (a == b)`	Are **a** and **b** equal?	Common mistake: **=** is used
`if (a != b)`	Are **a** and **b** not equal?	
`if (a <= b)`	Is **a** less than or equal to **b**?	Common mistake: **=<** is used
`if (a >= b)`	Is **a** greater than or equal to **b**?	Common mistake: **=>** is used
`if (a < b)`	Is **a** strictly less than **b**?	
`if (a > b)`	Is **a** strictly greater than **b**?	

Improved `printTicket()`

- Now the ticket only gets printed when there's sufficient balance!

```
public void printTicket() {
    if (balance >= price) {
        // There's enough money in the balance to buy a ticket!
        System.out.println("##########################");
        System.out.println("# The Puget Sound Line");
        System.out.println("# Ticket");
        System.out.println("# " + price + " dollars.");
        System.out.println("##########################");

        balance -= price;
        total += price;
    }
    else {
        // They must not have inserted enough money yet
        System.out.println("Fail: Still owe $" + (price - balance) + "!");
    }
}
```

Fun Exercise: Discounting Tickets

- Write a method `public void discount(int amt)`, which subtracts the given amount from the current price of a ticket.

- What can go wrong below?
  - Fix this code!

```
public void discount(int amt) {
    price -= amt;
}
```

- (Test it out with different inputs.)

price (before)	amt	price (after)
3	2	?
3	-2	?
3	10	?
3	-10	?
10	3	?
10	-3	?

Fun Exercise: Discounting Tickets

- Write a method `public void discount(int amt)`, which subtracts the given amount from the current price of a ticket.

- What can go wrong below?
  - Fix this code!

```
public void discount(int amt) {
    price -= amt;
}
```

- (Test it out with different inputs.)

price (before)	amt	price (after)	passed test?
3	2	1	Y
3	-2	5	N
3	10	-7	N
3	-10	13	N
10	3	7	Y
10	-3	13	N

Fun Exercises: Discounting Tickets

```
public void discount(int amt) {
    price -= amt;
}
```

- There could be two problems:

1. Discount **`amt`** could be negative!
   - (If user input **-6**, they probably meant to input **6**)
   - **Proposed fix:**
     - Negate the value of **`amt`** when this is the case
2. Discount **`amt`** could be more than the price of a ticket!
   - **Proposed fix:**
     - Set ticket price to 0
     - Alert user of their error

discount() Solution

```java
/**
 * Discounts the current price by the given amount
 * @param amt discounted amount
 */
public void discount(int amt) {
    // amt given as a negative. Negate it.
    if (amt < 0) {
        amt = -amt;
    }

    // Apply the discount optimistically
    price -= amt;

    if (price < 0) {
        // uh oh, price is negative, so the discount was too large
        System.out.println("Discount exceeds price. Price zeroed out.");
        price = 0;
    }
}
```

Outline

- Writing Our First Class: TicketMachine
  - Fields
  - Constructors
    - Parameters
  - Methods
    - Printing to screen
    - Local Variables
    - If-Statements
  - Using TicketMachines in Code (no more point and click!)
  - More practice with if-then-else
    - Logical operators
- Conclusion

Streamlining Your Tests

- Test by simulate a real-world scenario:
  - One ticket machine; tickets cost $5
  - Person pushes refund
  - Person inserts $6
  - Person prints ticket

  - Person pushes refund

- Sure, we can test classes using point-and-click, but isn't there a more efficient way?

Create a New "Tester" Class

- **Step 1:** Call the new class whatever you want, but I'll call it "Tester"
- **Step 2:** Create the following (special) method called **main()**:

```
public class Tester {

    public static void main(String[] args) {

        // testing algorithm goes in here

    }
}
```

- **Step 3:** Run the main method directly from BlueJ's project window

Testing Scenario

```
public class Tester {

    public static void main(String[] args) {
        // one ticket machine; tickets cost $5

        // person pushes refund

        // person inserts $6

        // person prints ticket

        // person pushes refund
    }
}
```

A Quick "HOW TO" Guide!

- Syntax to create a new Object

```
ClassName variable-name = new ClassName(list-of-arguments);
```

  - Arguments are values you want to be input into the parameters.

- The example below creates 2 ticket machines called **t1** and **t2**:

```
// This ticket machine dispenses tickets of random cost
TicketMachine t1 = new TicketMachine();

// Tickets in this machine cost $5
TicketMachine t2 = new TicketMachine(5);
```

A Quick "HOW TO" Guide!

- Syntax to *call a method* on an Object
  - Called "dot notation"

```
variable-name.method-name(list-of-arguments);
```

- Examples of method calls:

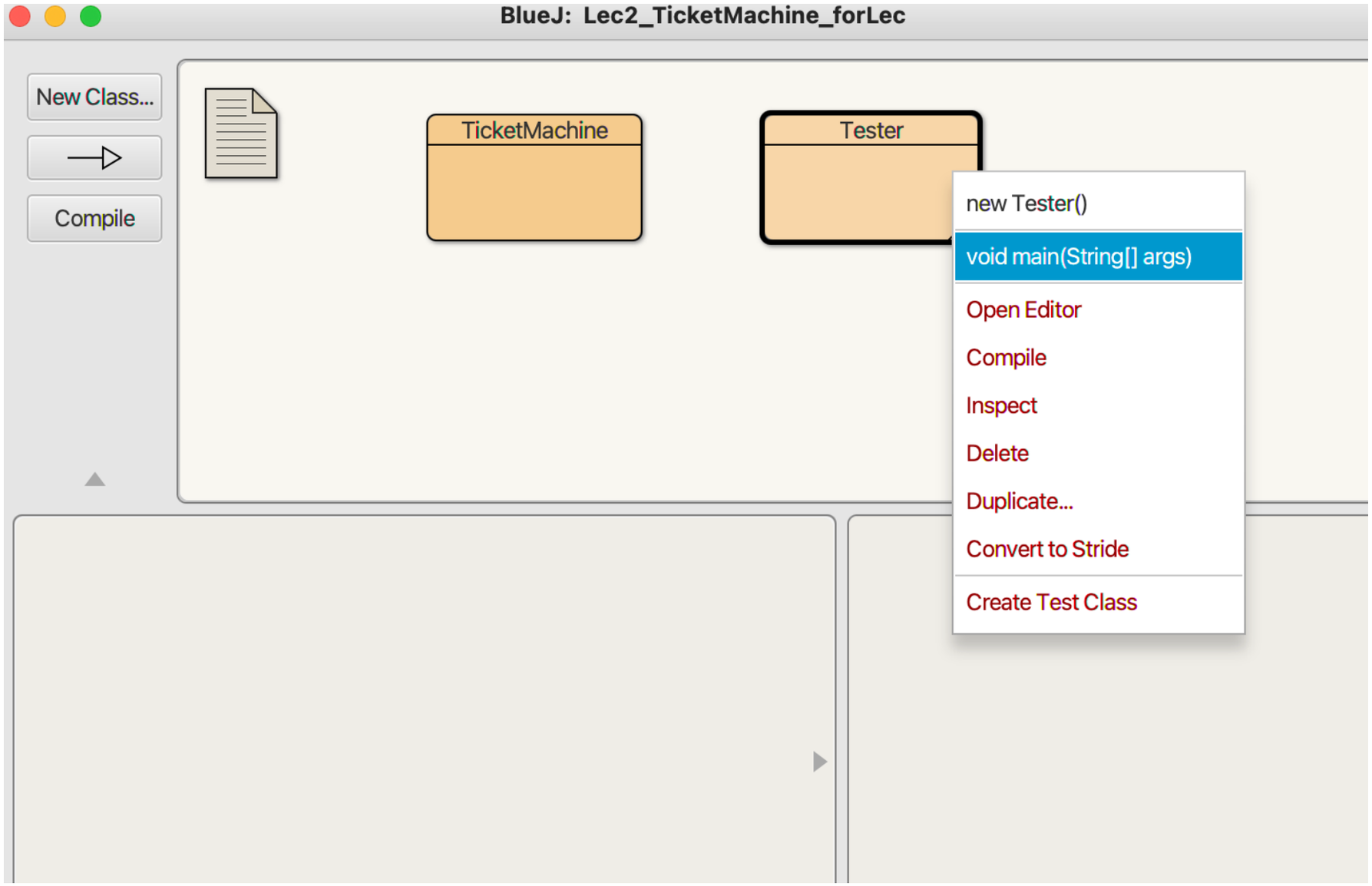
```
// This ticket machine dispenses tickets of random cost
TicketMachine t1 = new TicketMachine();

// tickets in this machine below cost $5
TicketMachine t2 = new TicketMachine(5);

// call a method here
t2.insertMoney(50);
```

Example Testing Scenario

```
public class Tester {

    public static void main(String[] args) {
        // Construct a new ticket machine; tickets cost $5
        TicketMachine tm = new TicketMachine(5);

        // person pushes refund
        tm.refundBalance();

        // person inserts $6
        tm.insertMoney(6);

        // person prints ticket
        tm.printTicket();

        // person pushes refund
        tm.refundBalance();
    }
}
```

Now Run the Script!

- Right click on Tester, and choose **main(String[] args)** to run it!



Outline

- Writing Our First Class: TicketMachine
  - Instance Variables
  - Constructors
    - Parameters
  - Methods
    - Printing to screen
    - Local Variables
    - If-Statements
  - Using TicketMachines in Code (no more point and click!)
  - More practice with if-then-else (and Flowcharts)
    - Logical operators
- Conclusion

Selecting from Multiple Alternatives

- Write a method called `weather()` that prints out a message given some temperature in Fahrenheit, `t`

Temperature	Output message
Above 95	"Blazing"
Above 80	"Hot"
Above 50	"Pleasant"
At or below 50	"Cool"

- What's ***wrong*** with the code below?

```
public void weather(double t) {

    if (t > 95) {
        System.out.println("Blazing");
    }

    if (t > 80) {
        System.out.println("Hot");
    }

    if (t > 50) {
        System.out.println("Pleasant");
    }

    if (t <= 50) {
        System.out.println("Cool");
    }
}
```

Multiple Alternatives: Else-If Statements

- *Else-if Statements* improve the readability of multiple alternatives!
  - **Important:** When a condition succeeds, all subsequent conditions are skipped.

```
if (condition-1) {
   // do this if only condition-1 is true
}
else if (condition-2) {
   // do this if only condition-2 is true
}
else if (condition-3) {
   // do this if only condition-3 is true
}
else if (condition-4) {
   // do this if only condition-4 is true
}
else {
   // do this if all the conditions above fail
}
```

Multiple Alternatives: Else-If Statements

- Example use of Else-if statements to select one option among multiple alternatives!

- The nested code from before can be vastly simplified as follows:

```
public void weather(double t) {

    if (t > 95) {
       System.out.println("Blazing");
    }
    else if (t > 80) {
       System.out.println("Hot");
    }
    else if (t > 50) {
       System.out.println("Pleasant");
    }
    else {
      System.out.println("Cool");
    }
}
```

Outline

- Writing Our First Class: TicketMachine
  - Fields
  - Constructors
    - Parameters
  - Methods
    - Printing to screen
    - Local Variables
    - If-Statements
    - More practice with if-then-else
    - Boolean (Logical) Operators
- Conclusion

Boolean Operators: Combining Conditionals

- We can combine multiple boolean expressions.

- The ***and*** operator: `if (condition1 && condition2 && ...)`
  - Triggers only if ***all*** conditions are **true**

- The ***or*** operator: `if (condition1 || condition2 || ...)`
  - Triggers if ***any*** of the conditions are **true**

- The ***not*** operator: `if (!condition)`
  - Triggers if the negation of the given condition is **true**

Example: Use of &&

- Write a method:
  - `increasingOrder` that inputs 3 integers x, y, and z, and *returns* true if they are given in increasing order, and returns false otherwise.

```
/**
 * This method determines if its inputs are given in increasing order
 * @param x
 * @param y
 * @param z
 * @return true if x < y < z, and false otherwise
 */

public ____TODO____ increasingOrder(____TODO____) {


      //____TODO____

}
```

Solution

- Write a method:
  - `increasingOrder` that inputs 3 integers x, y, and z, and *returns* true if they are given in increasing order, and returns false otherwise.

```
/**
 * This method determines if its inputs are given in increasing order
 * @param x
 * @param y
 * @param z
 * @return true if x < y < z, and false otherwise
 */

public boolean increasingOrder(int x, int y, int z) {

   if (x < y && y < z) {
      return true;
   }
   else {
      return false;
   }
}
```

Solution (Simplified)

- This version accomplishes the same thing, but is simplified to be in non-redundant form:

```
/**
 * This method determines if its inputs are given in increasing order
 * @param x
 * @param y
 * @param z
 * @return true if x < y < z, and false otherwise
 */

public boolean increasingOrder(int x, int y, int z) {
   return (x < y && y < z);
}
```

Another Example

- Use boolean operator(s). No nested if-statements necessary
- Write a method
  - `isLiquid` that inputs the temperature (Celsius) of water and *returns* whether the water is in liquid state.
    - Water is a *solid* when temperature is equal or below 0 degree celsius
    - Water is a *gas* when temperature is equal or above 100 degreecelsius

```
public _______ isLiquid(________) {


   // Your turn!


}
```

IsLiquid? (Soln)

- Solutions (all correct)
  - Thew work:

```
public boolean isLiquid(int temp) {
   if (temp > 0 && temp < 100) {
      return true;
   }
   return false;
}
```

```
public boolean isLiquid(int temp) {
   if (temp <= 0 || temp >= 100) {
      return false;
   }
   return true;
}
```

  - Simplified forms:

```
public boolean isLiquid(int temp) {
   return (temp > 0 && temp < 100);
}
```

```
public boolean isLiquid(int temp) {
   return !(temp <= 0 || temp >= 100);
}
```

More Practice: Vowel?

- Write two methods:
  - `isVowel()` that inputs a character (**`char`** data type) *returns* **`true`** if it's a vowel (a,e,i,o,u). Otherwise, it returns **`false`**.
  - `isConsonant()` that inputs a character (**`char`** data type) *returns* **`true`** if it's a consonant (b,c,d,f,...,y,z). Otherwise, it returns **`false`**.

```
public _______ isVowel(________) {

    // TODO
}

public _______ isConsonant(________) {

    // TODO
}
```

Solution: Vowel?

```java
public boolean isVowel(char letter) {
    if ('a' == letter || 'e' == letter || 'i' == letter || 'o' == letter || 'u' == letter) {
        return true;
    }
    return false;
}


public boolean isConsonant(char letter) {
    // if it's not a vowel, then it's a consonant!
    if (isVowel(letter) == false) {
        return true;
    }
    return false;
}
```

- Simplified:

```java
public boolean isVowel(char letter) {
    return ('a' == letter || 'e' == letter || 'i' == letter || 'o' == letter || 'u' == letter);
}


public boolean isConsonant(char letter) {
     return !isVowel(letter);
}
```

Example: Closed or Open?

- Suppose the office is closed
  - Between hours 2-8 on weekdays
  - And on all hours on weekends

- Write the following method *prints* indicating if the office is open:

- Two inputs:
  - **weekend** is given as a true/false value
  - **hour** is given as a number between 0 and 23

```
public void isOpen(____TODO____) {

    // TODO

}
```

Closed or Open? (Soln)

- Suppose the office is closed
  - Between hours 2-8 on weekdays
  - And on all hours on weekends

```
public void isOpen(boolean weekend, int hour) {
   if (!weekend && (hour < 2 || hour > 8)) {
      System.out.println("Open");
   }
   else {
      System.out.println("Closed");
   }
}
```