

→

→



# ADMINISTRATION

- **Assignment #1 due now**
  - Code and sample output in Canvas
- **Class TA**
  - Rishabh Sood – [sood.r@northeastern.edu](mailto:sood.r@northeastern.edu)
  - MSTeams connection – link available via Canvas announcement

# THE LECTURE

- **Recap**
- **Data Structures**
  - Arrays
  - Classes, new, 'dot'
- **Shifting to OO**
  - Classes and Objects
  - Strings
- **Assignment #2a**

# RECAP

# ? OPERATOR

As a convenience to replace an if-else block of form:

*if (condition)*

*var = expression1;*

*else*

*var = expression2;*

Use the ? Operator form:

*var = (condition) ? expression1 : expression2;*

Example:

*maxLen = (lookForward < 5) ? 12 : minLength-1;*

# PROCEDURAL PROGRAMMING

- Software constructed by a collection of procedure calls (i.e., methods, subroutines, functions)

- Functional

- Uses parameters for input
  - Returns the same results every time

```
int result = add(5, 6);
```

- Imperative

- Uses 'implied' state or data
  - Results may vary depending on the current state

```
setPenColor(BLUE);  
drawText("Hello");
```

**ARRAYS**

# **DATA STRUCTURES**

# ARRAYS



# ONE-DIMENSIONAL ARRAYS

A one-dimensional array is declared using the form:

*type array-name***[ ]** = new *type***[size]**;

Examples:

```
int samples[] = new int[10];
```

```
for (int i = 0; i < samples.length; i++)  
    samples[i] = i;
```

# TWO-DIMENSIONAL ARRAYS

Two-dimensional arrays are specified with the form:

*type array-name***[ ][ ]** = new *type***[rowSize][columnSize]**;

Examples:

```
int sqrs[][] = { {1, 1},  
                 {2, 4},  
                 {3, 9} };
```

```
// 5 rows, 3 columns  
int num[][] = new int[5][3];
```

# FOR-EACH LOOPS

Arrays be be accessed using a for-each loop with the form:

*for (type itr-var : collection) statement;*

Example:

```
int nums[] = { 1,2,3,4,5,6,7,8,9,10 };
```

```
int sum = 0;
```

```
for (int x : nums)
```

```
    sum += x; // same as 'sum = sum + x;'
```

# CREATING NEW DATA TYPES

Multiple data elements may be grouped together to form a new complex data type

A Java class represents the plans for building a new data type

C:

```
struct Vehicle {  
    int passengers;  
    int fuelCap;  
    double mpg;  
}
```

Java:

```
class Vehicle {  
    int passengers;  
    int fuelCap;  
    double mpg;  
}
```

Class Name

Member  
variables

# THE 'NEW' COMMAND

A class definition can be used to generate an instance of a class, creating a new object.

Vehicle.java

```
class Vehicle {  
    int passengers;  
    int fuelCap;  
    double mpg;  
}
```

```
class VehicleTest {  
    public static void main(String args[]) {  
        Vehicle minivan = new Vehicle();  
        Vehicle sportscar = new Vehicle();  
    }  
}
```

# THE DOT (.) OPERATOR

Member values in a class may be assigned using the dot operator

```
minivan.passengers = 8;
```

```
minivan.fuelCap = 15;
```

```
minivan.mpg = 28.0;
```

```
sportscar.passengers = 2;
```


```
sportscar.fuelCap = minivan.fuelCap;
```

```
// Both vehicles have a fuelCap value of 15
```

```
sportscar.mpg=20.0;
```

# MEMBERS ASSIGN VALUES

`minivan.mpg = 28.0;`




A blue curved arrow points from the code `minivan.mpg = 28.0;` to the `minivan` object. A grey curved arrow points from the `mpg` property in the `minivan` object to the value `28.0`.

passengers	8
fuelCap	15
mpg	28.0

`.mpg`

`sportscar.fuelCap = 15;`



A blue curved arrow points from the code `sportscar.fuelCap = 15;` to the `sportscar` object. A grey curved arrow points from the `fuelCap` property in the `sportscar` object to the value `15`.

passengers	2
fuelCap	15
mpg	20.0

`.fuelCap`

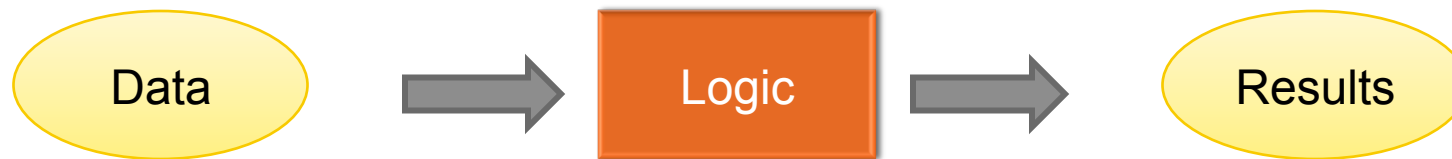
# SHIFTING TO O-O



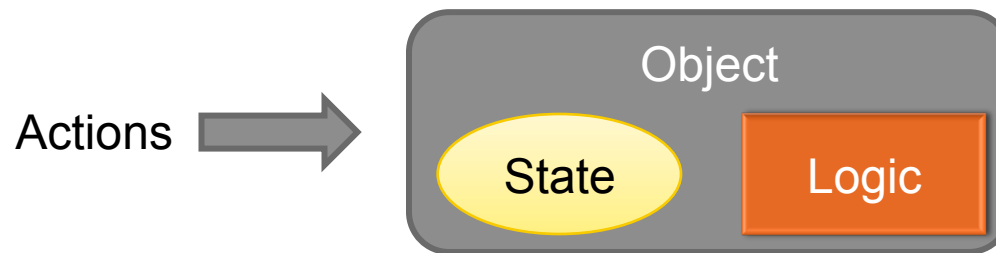
# OBJECTS

# OBJECT-ORIENTED

**Historical model: Data is fed into actions for processing (logic) to produce/output results**



**The Object-Oriented model changes that view, and instead says that there are objects that need manipulation.**



**An Object is an entity that has state and performs actions or has behavior**

# DEFINING OBJECTS

## THE CLASS FORM:

- A class may be specified using the following form:

```
class classname {  
    // instance variables  
    type var1;  
    type var2;  
  
    // declare methods  
    ret-type method1(parameters) {  
        // body of method  
    }  
    ret-type method2(parameters) {  
        // body of method  
    }  
}
```

# CONSTRUCTORS

- Each class may specify methods that will be called when a class instance is generated
- These special methods are called Constructors, and share the same name as the class
- Constructors do not specify a return type (if it did, it would just be a 'method')

```
class XClass {  
    int x;  
  
    XClass() {  
        x = 10;  
    }  
}
```

# CONSTRUCTORS (CONT.)

- Constructors can be used to initialize an object when it is created

```
class XClass {  
    int x;  
  
    XClass() { // default constructor  
        x = 10;  
    }  
    XClass(int val) { // constructor with input parameters  
        x = val;  
    }  
}
```

- The following statements would create an instance of XClass, and initialize x to 10:

```
XClass myX1 = new XClass();  
XClass myX2 = new XClass(10);
```

# THE 'THIS' KEYWORD

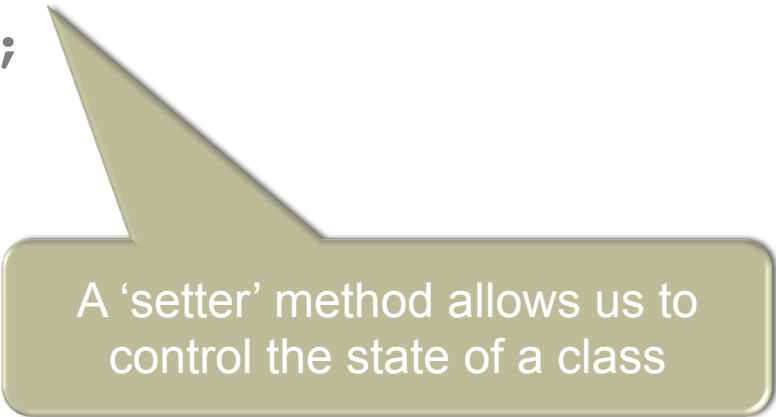
- All method calls are automatically passed a reference value for the invoking object. This reference is called **this**.
- The **this** reference is handy for passing your reference to other objects

```
void addUsToList() {  
    vehicleServiceList.addVehicle(this);  
}
```

# 'THIS' (CONT.)

- The this reference can be use to remove class/method variable ambiguity

```
class Vehicle {  
    int seats;  
  
    //...  
  
    void setSeats(int seats) {  
        this.seats = seats;  
    }  
}
```



A 'setter' method allows us to control the state of a class

# STRINGS



# STRINGS

```
String name = "Mark Munson";
```

- In many other languages, a **String** is just an array of characters
- In Java, a **String** is an Object (holding an array of characters)
- Every time you use a string literal, you are using a **String**

```
System.out.println("Java strings are objects.");  
String str0 = "Java strings are powerful."  
String str1 = new String("They are constructed many ways");  
String str2 = new String(str1);
```

# STRINGS (CONT.)

- **Strings are immutable**
  - Once made, they cannot be altered
  - Strings with the same text may point to a common array of characters

- **Strings may be concatenated together**

```
String title = "First part" + " second part";
```

- **Strings may be defined in arrays**

```
String strs[] = { "My", "name", "is", "Mark" };
```

# STRINGS (CONT.)

- **Strings may be concatenated together**

```
String title = "First part" + " second part";
```

- **Strings may be defined in arrays**

```
String strs[] = { "My", "name", "is", "Mark" };
```

- **When objects are concatenated with a string, the toString() methods is called on the object to create a String**

```
Vehicle truck = new Vehicle();
```

```
String intro = "My vehicle is a " + truck;
```

Is the same as:

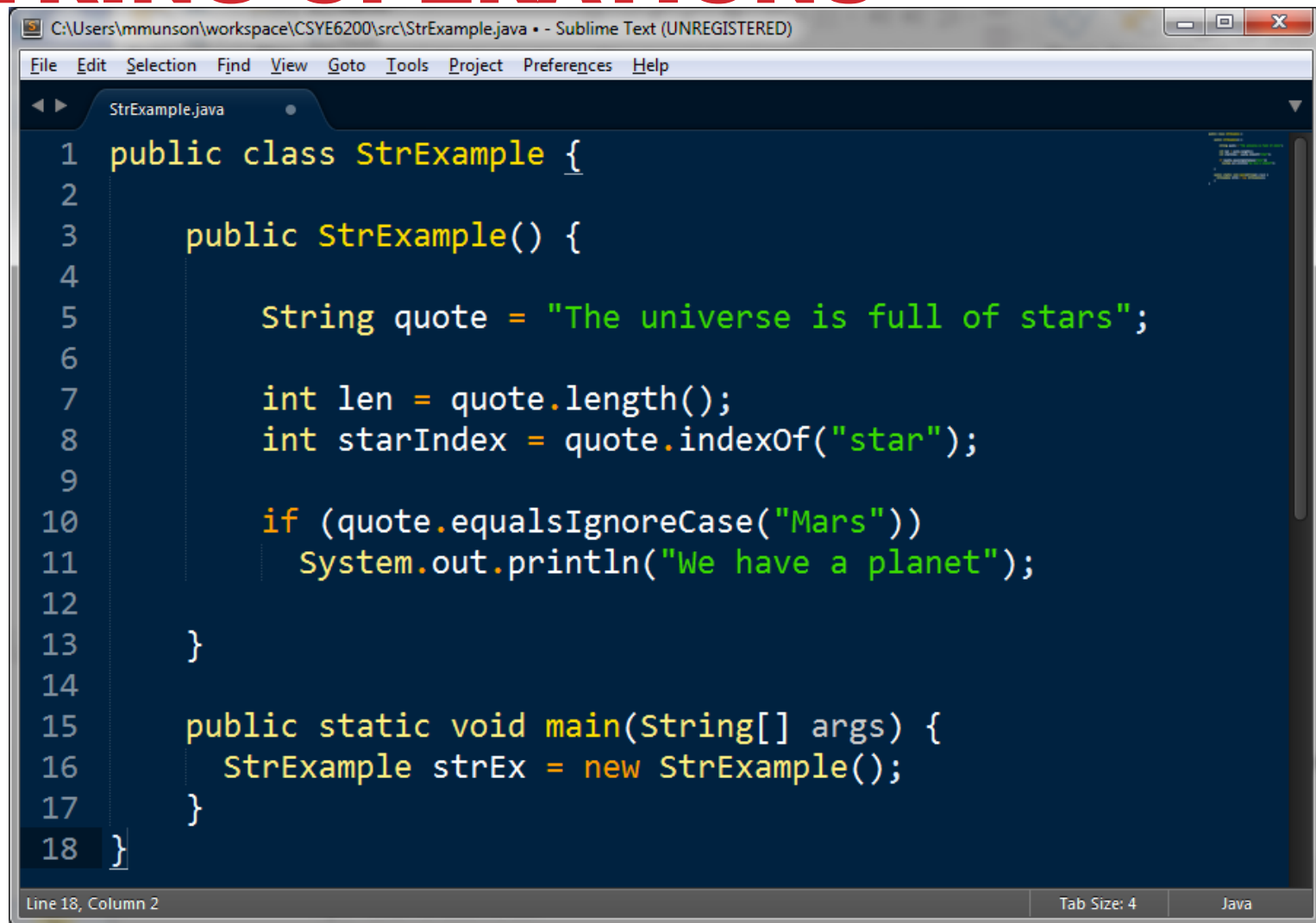
```
String intro = "My vehicle is a "+ truck.toString();
```

# STRINGS IN SWITCH

- As of Java 7, Strings may be used in a switch statements

```
String command = "start";
switch(command) {
    case "start":
        // statement;
        break;
    case "stop":
        // statement;
        break;
    default:
    case "reset":
        // statement;
        break;
}
```

# STRING OPERATIONS



The screenshot shows a Sublime Text editor window with the title bar "C:\Users\mmunson\workspace\CSYE6200\src\StrExample.java • - Sublime Text (UNREGISTERED)". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The editor displays a Java file named StrExample.java with the following code:

```
1 public class StrExample {
2
3     public StrExample() {
4
5         String quote = "The universe is full of stars";
6
7         int len = quote.length();
8         int starIndex = quote.indexOf("star");
9
10        if (quote.equalsIgnoreCase("Mars"))
11            System.out.println("We have a planet");
12    }
13
14    public static void main(String[] args) {
15        StrExample strEx = new StrExample();
16    }
17 }
18 }
```

The status bar at the bottom indicates "Line 18, Column 2", "Tab Size: 4", and "Java".

# NEXT WEEK / ASSIGNMENT #2A

JABG:  
Read Ch. 6 & 7  
(Methods, Overloading, Static,  
Recursion, Inheritance)

**Assignment:** Auto Rental Agency – Due Tuesday, **February 16<sup>th</sup>, 11:59 pm EST**

- **Write a `Vehicle.java` class and a `VehicleTest.java` class**
- **Use the sample starter code (`CSYE6200Assign2.zip`) which will be uploaded to the course material site. Please fill in your name and NUID number.**
- **To the `Vehicle` class**
  - add Strings for both the make and model (i.e. make: Volvo, model: S80)
  - Add a constructor that sets the make and model, along with the other instance variables
  - Add a model year
  - Add a method to calculate the vehicle range
- **In the `VehicleTest` program, use the ‘new’ operator with your `Vehicle` constructor to generate three instances of different vehicle objects.**
- **Add a method to print an attractive display of the vehicle data including the range**
- **Submit your source code to Canvas as .java files. Include a copy of your program’s output captured in a text file.**