Classes & Objects
Data Structures

Thus far, all of our data has been stored in variables or arrays

- one variable holds one piece of data
- one array holds multiple pieces of data of the same datatype

_Data structures_ enable our programs to organize our data in more efficient, sensible ways

We'll see three types of data structures this semester

- variables (all semester)
- arrays (remainder of semester)
- classes (this week)
Classes

Allows us to group together pieces of data that define a real world concept even if they are of different datatypes!

  e.g., a professor is made up of a first/last name, courses they teach...

A *class* provides a definition of what pieces of data define a real world concept

An *object* defines a particular *instance* of that class
UWL as Object-Oriented Data

Professor
(name, list of classes, office)

class

objects

Allie Sauppé
CS120, CS364
Wing 214

Marty Allen
CS120, CS227
Wing 210

Elliot Forbes
CS272, CS370
Wing 219

Jason Sauppe
CS225, CS270
Wing 207

Sam Foley
CT100, CS441
Wing 220

Tom Gendreau
CS340, CS470
Wing 211
Components of Classes

Identifier

name of the class
should be singular, start with a capital letter (e.g., Professor, Student)

Attributes

data that defines every object of that class type

Methods

define the actions that can be taken with objects of that class type
Object-oriented programs are comprised of **objects** from **multiple** classes **interacting**.
Java is made up of thousands of classes. But, we can create our own classes for our needs too.
public class Professor {
    private String firstName;
    private String lastName;
    private String dept;
    private Course[] courses;

    public Professor(String fn, String ln) {
        this.firstName = fn;
        this.lastName = ln;
    }

    public String getDept() {
        return dept;
    }

    public void setDept(String dept) {
        this.dept = dept;
    }
}

only part of the class (missing many details)
Classes in Java: Identifier

public class Professor {

}

Name of the class
Should be singular
Should start with a capital letter (e.g., Professor, Student)
Classes in Java: Attributes

Data that defines every object of that class type

Variable declarations at a minimum can also initialize/instantiate if needed

Also referred to as *global variables*

have scope throughout the class

New concept: visibility

public, private, protected

```java
public
{
  private String firstName;
  private String lastName;
  private String dept;
  private Course[] courses;
}
```
Classes in Java: Methods

Define the actions that can be taken with objects of that class type

Work like methods from last week

Key differences

- lack of static keyword
- use of this keyword
- no main method
Classes in Java: Constructor Method

Method to create (*instantiate*) an object of this class type

Named the same as the class

Lacks a return type

Seen these throughout the semester

```java
public Professor(String fn, String ln) {
    this.firstName = fn;
    this.lastName = ln;
}
```
Example: Constructor

```java
public class UWL {
    public static void main(String[] args) {
        Professor aSauppe = new Professor("Sauppe", "Allie");
    }
}

public class Professor {

    private String firstName;
    private String lastName;

    public Professor(String ln, String fn) {
        firstName = fn;
        lastName = ln;
    }
}
```
Example: Constructor

```java
public class UWL {
    public static void main(String[] args) {
        Professor aSauppe = new Professor("Sauppe", "Allie");
    }
}
```

```java
public
private
private
public
}
```
Example: Constructor

```java
public class Professor {
    private String firstName;
    private String lastName;

    public Professor(String fn, String ln) {
        firstName = fn;
        lastName = ln;
    }
}
```

Memory:
- fn (String) = "Sauppe"
- ln (String) = "Allie"

Usage:
```java
new Professor("Sauppe", "Allie");
```
Example: Constructor

```java
public class Professor {
    private String firstName;
    private String lastName;

    public Professor(String ln, String fn) {
        firstName = fn;
        lastName = ln;
        // Constructing an object
        new Professor("Sauppe", "Allie");
    }
}
```

```
memory

<table>
<thead>
<tr>
<th>fn (String)</th>
<th>ln (String)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Sauppe&quot;</td>
<td>&quot;Allie&quot;</td>
</tr>
</tbody>
</table>

(Professor)

firstName = "Allie"
lastName = "Sauppe"
```
Example: Constructor

```java
public class UWL {
    public static void main(String[] args) {
        Professor aSauppe = new Professor("Sauppe", "Allie");
    }
}
```

Example:

```java
public class UWL {
    public static void main(String[] args) {
        Professor aSauppe = new Professor("Sauppe", "Allie");
    }
}
```

Example:

```java
public class UWL {
    public static void main(String[] args) {
        Professor aSauppe = new Professor("Sauppe", "Allie");
    }
}
```
Classes & Methods

Methods are *always* affiliated with a class.

Available data:
- variables declared in the method
- parameters
- global variables for that class
calculateAge: Before

What we saw previously...

```java
public static int calculateAge(int bYear, int bMonth, int bDay, int tYear, int tMonth, int tDay) {
    int age = tYear - bYear;

    if (bMonth > tMonth || (bMonth == tMonth && bDay > tDay)) {
        age--;
    }

    return age;
}
```
public class Student {
    private String firstName;
    private String lastName;
    private int bYear;
    private int bMonth;
    private int bDay;

    // ...

    public int calculateAge(int tYear, int tMonth, int tDay) {
        int age = tYear - bYear;

        if (bMonth > tMonth || (bMonth == tMonth && bDay > tDay)) {
            age--;
        }

        return age;
    }
}
public class UWL {
    public static void main(String[] args) {
        // ...
        // students have already been instantiated
        int jamesAge = james.calculateAge(2017, 11, 7);
        int elizaAge = eliza.calculateAge(2017, 11, 7);
    }
}

public class Student {
    // ...
    public int calculateAge(int tYear, int tMonth, int tDay) {
        int age = tYear - bYear;
        if(bMonth > tMonth || (bMonth == tMonth && bDay > tDay)) {
            age--;
        }
        return age;
    }
}
Visibility

Used to control access to classes, methods, and attributes

Three options

  public: can be accessed from any class
  private: can only be accessed from its own class
  protected: we'll get to this later

Visibility applies to classes, method, and global variables

  public class Professor
  public static void printArray(char[] arr)
  private String firstName
Visibility Rules of Thumb

Classes are usually public
    tend to only be useful to us if they can be accessed from other classes
Attributes are usually private
    don’t want people to change them at will
    forces change through methods, which provide guarantees
Methods are most likely public, but private is also common
    public methods used to work with objects of that type
    private methods used to help internal class functionality
Getter and Setter Methods

Since attributes are usually private, need some way to access them

*Getter methods* get the value of an attribute

*Setter methods* set the value of an attribute

  can be used to ensure the attribute is only set to sensible values

  e.g., only possible values for birth month are 1-12

Example for firstName attribute

  public String getFirstName()

  public void setFirstName(String fn)
final Keyword

Modifier used for classes, methods, and variables

we'll only talk about variables

Variables with the final keyword can only be assigned a value once

Examples from Math class

Math.PI (3.14159…)
Math.E (2.71828…)

Final variables are written in all uppercase, with underscores for spaces

e.g., MAX_COURSE_LOAD
Static vs Non-Static Methods

The *static* keyword controls whether a resource (e.g., method, variable) belongs to the *class* or an *object* of that class type

- **static**: do not need to have instantiated an object of that class type to use it
- **non-static**: must have an object instantiated of that class type

Overarching question: Do I need to know one or more attribute values from an object to use this?

- yes? **non-static**
  - default should be non-static
- no? **static**
Static Rules of Thumb

Generally, methods/variables will be non-static
  conforms to object-oriented principles
Static methods can only access static attributes
  non-static methods can access all attributes
Examples of static methods from Java:
  everything from the Math class
  Math.pow(double x, int y)
  Math.max(double x, double y)
How to Call Methods

Is the method I want to call static?

- **yes**
  - `<Class>.<methodName>(<args>)`
  - or
  - `<methodName>(<args>)`
  - will assume the class you are currently in

- **no**
  - `<object>.<methodName>(<args>)`
  - or
  - `<methodName>(<args>)`
  - must already be in the class; will assume the object you are currently using
Steps to Creating a New Class

1. Class name

2. Attributes
   - name, type, visibility, initialization/instantiation?

3. Constructor method
   - parameters come from attributes

4. Other methods
   - getters/setters, methods specified in requirements
Easy way to represent basic components of a class (name, attributes, methods)

Part of *unified modeling language* (*UML*)

*used to communicate structure of programs*

**Visibility prefaces identifier**

+ for public
– for private
# for protected

Static attributes/methods are underlined
Attributes list type after colon
Methods list only parameter types
Return type appears after method, prefaced with a colon
constructor will not list a return type
list void if no return type

Class Diagram

```plaintext
Student

- firstName : String
- lastName : String
- birthYear : int
- birthMonth : int
- birthDay : int

+ Student(String, String, int, int, int)
+ getFirstName() : String
+ setFirstName(String) : void
+ calculateAge(int, int, int) : int
```
Object Diagram

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstName = &quot;Jimmy&quot;</td>
</tr>
<tr>
<td>lastName = &quot;Gordon&quot;</td>
</tr>
<tr>
<td>birthYear = 1994</td>
</tr>
<tr>
<td>birthMonth = 4</td>
</tr>
<tr>
<td>birthDay = 8</td>
</tr>
</tbody>
</table>

Used to identify current state of object
Lists current values for each attribute
Does not list methods
Object Tracing

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
ma = as;

System.out.println(as.getFirstName() + " " + ma.getFirstName());
```
Object Tracing

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
ma = as;
System.out.println(as.getFirstName() + " " + ma.getFirstName());
```

no longer any variable referring to this object!
(orphaned object)
Object Tracing

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
ma = as;

System.out.println(as.getFirstName() + " " + ma.getFirstName());
```

```
Professor
firstName = "Allie"
lastName = "Sauppe"

Professor
firstName = "Marty"
lastName = "Allen"
```

cannot reestablish a reference to this object; collected by Java’s garbage collector
**Object Tracing**

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
ma = as;

System.out.println(as.getFirstName() + " " + ma.getFirstName());

Allie Allie
```
Object Tracing

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
Professor temp;

temp = as;
    as = ma;
    ma = temp;
    temp = null;
System.out.println(as.getFirstName() + " " + ma.getFirstName());
```

**Professor**
- `firstName = "Allie"
- `lastName = "Sauppe"

**Professor**
- `firstName = "Marty"
- `lastName = "Allen"

**temp** → null
Object Tracing

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
Professor temp;

temp = as;
> as = ma;
  ma = temp;
  temp = null;
System.out.println(as.getFirstName() + " " + ma.getFirstName());
```

```
Professor
firstName = "Allie"
lastName = "Sauppe"

Professor
firstName = "Marty"
lastName = "Allen"
```

null
Object Tracing

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");

Professor temp;

temp = as;
as = ma;
> ma = temp;
temp = null;
System.out.println(as.getFirstName() + " " + ma.getFirstName());
```

```
Professor
firstName = "Allie"
lastName = "Sauppe"

Professor
firstName = "Marty"
lastName = "Allen"

temp
null
```
Object Tracing

Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
Professor temp;

temp = as;
as = ma;
ma = temp;
> temp = null;
System.out.println(as.getFirstName() + " " + ma.getFirstName());

Professor
firstName = "Allie"
lastName = "Sauppe"

Professor
firstName = "Marty"
lastName = "Allen"

null
Object Tracing

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
Professor temp;

temp = as;
as = ma;
ma = temp;
temp = null;
System.out.println(as.getFirstName() + " " + ma.getFirstName());
```
Object Tracing

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
Professor temp;
temp = as;
as = ma;
ama = temp;
temp = null;
System.out.println(as.getFirstName() + ", " + ma.getFirstName());
Marty Allie
```
always treat variables of a class type and the objects they refer to as two separate entities
Object Tracing

Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
Professor temp;

temp = as;
as = ma;
ma = temp;
temp = null;
System.out.println(as.getFirstName() + " " + ma.getFirstName());
Object Tracing

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
Professor temp;
temp = as;
> as = ma;
  ma = temp;
  temp = null;
System.out.println(as.getFirstName() + 
  " " + ma.getFirstName());
```

Diagram:
- Professor
  - FirstName: "Allie"
  - LastName: "Sauppe"
- Professor
  - FirstName: "Marty"
  - LastName: "Allen"
- null

Note:
- `as` and `ma` are initially set to Professor objects with different names.
- `temp` is used to swap their references.
- The final `println` will show the names of `as` and `ma` after the swap.
Object Tracing

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
Professor temp;

temp = as;
as = ma;
> ma = temp;
temp = null;
System.out.println(as.getFirstName() + " " + ma.getFirstName());
```

```
Professor

firstName = "Allie"
lastName = "Sauppe"

Professor

firstName = "Marty"
lastName = "Allen"

null
```
Professor as = new Professor("Sauppe", "Allie");

Professor ma = new Professor("Allen", "Marty");

Professor temp;

temp = as;
as = ma;
ma = temp;
>temp = null;
System.out.println(as.getFirstName() + " " + ma.getFirstName());
Object Tracing

Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
Professor temp;

temp = as;
as = ma;
ama = temp;
temp = null;
> System.out.println(as.firstName() + " " + ma.firstName());
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
Professor temp;
temp = as;
as = ma;
ama = temp;
temp = null;
System.out.println(as.getFirstName() + " " + ma.getFirstName());
Marty Allie
Objects/Arrays & Methods

```java
> int[] array = {3, 1, 2, 5, 9};
> swap(array, 1, 4);
```

```java
public static
    
    
arr
}
```
Objects/Arrays & Methods

```java
int
> swap(array, 1, 4);

public static void swap(int[] arr, int i1, int i2) {
    int temp = arr[i1];
    arr[i1] = arr[i2];
    arr[i2] = temp;
}
```
int
>
> swap(array, 1, 4);

```java
public static void swap(int[] arr, int i1, int i2) {
    int temp = arr[i1];
    arr[i1] = arr[i2];
    arr[i2] = temp;
}
```

array
arr
int[]
3 1 2 5 9
temp (int)
1
Objects/Arrays & Methods

```java
int
> swap(array, 1, 4);

public static void swap(int[] arr, int i1, int i2) {
    int temp = arr[i1];
    arr[i1] = arr[i2];
    arr[i2] = temp;
}
```
int

> swap(array, 1, 4);

public static void swap(int[] arr, int i1, int i2) {
    int temp = arr[i1];
    arr[i1] = arr[i2];
    arr[i2] = temp;
} >

array

arr

int[]

3 9 2 5 1

temp (int)

1
Objects/Arrays & Methods

```java
int[] array = {3, 1, 2, 5, 9};
swap(array, 1, 4);
```

```
public static
{
    arr
}
```
this Keyword

Used to refer to the object we are currently using

  e.g., this object

Can be used just like any other object
Object Tracing With Methods

Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
as.renameProf("Allison");
ma.renameProf("Martin");

public void renameProf(String newName) {
    this.firstName = newName;
}
Object Tracing With Methods

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
as.renameProf("Allison");

ma.renameProf("Martin");
```

Professor

<table>
<thead>
<tr>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Allie&quot;</td>
<td>&quot;Sauppe&quot;</td>
</tr>
</tbody>
</table>

Professor

<table>
<thead>
<tr>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Marty&quot;</td>
<td>&quot;Allen&quot;</td>
</tr>
</tbody>
</table>
Object Tracing With Methods

Professor
Professor
> as.renameProf("Allison");

public void renameProf(String newName) {
    > this.firstName = newName;
}

Professor
firstName = "Allie"
lastName = "Sauppe"
public void renameProf(String newName) {
    this.firstName = newName;
}
Object Tracing With Methods

```
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
> as.renameProf("Allison");
> ma.renameProf("Martin");
```

```
public
}
```
Object Tracing With Methods

Professor
Professor

as

> ma.renameProf("Martin");

public void renameProf(String newName) {
    > this.firstName = newName;
}

Professor

firstName
lastName

as

ma

this

Professor

firstName = "Marty"
lastName = "Allen"
Object Tracing With Methods

Professor

Professor

as

> ma.renameProf("Martin");

public void renameProf(String newName) {
    this.firstName = newName;
}>

Professor

firstName = "Martin"
lastName = "Allen"

Professor

firstName = "Martin"
lastName = "Allen"
Object Tracing With Methods

```java
Professor as = new Professor("Sauppe", "Allie");
Professor ma = new Professor("Allen", "Marty");
as.renameProf("Allison");
ma.renameProf("Martin");

public void renameProf(String newName) {
    this.firstName = newName;
}
```