Announcements

- Program 2 due Monday

- Midterm in one week
  - Moved to SN014 (next door)
  - Will cover up to yesterday (Ch. 1–4)

- Future office hours moved to FB331
Questions?
Today in COMP 110

- Briefly go over Strings and Loops Worksheet
- Classes
num++ vs. ++num

- num++ is NOT num + 1
- num++ does num = num + 1
- So does ++num, BUT, there is a difference

```java
int num1 = 5;
System.out.println(num1++);
// outputs num1 (5), then increments num1

int num2 = 5;
System.out.println(++num2);
// increments num2, then outputs num2 (6)
```
Classes and Objects

- Java programs (and programs in other object-oriented programming languages) consist of objects of various class types

- Objects can represent objects in the real world
  - Automobiles, houses, employee records

- Or abstract concepts
  - Colors, shapes, words
A class is the definition of a kind of object
  ◦ A blueprint for constructing specific objects

**Class Name:** Automobile

**Data:**
  amount of fuel
  speed
  license plate

**Methods (actions):**
  accelerate:
    **How:** Press on gas pedal.
  decelerate:
    **How:** Press on brake pedal.
Objects, Instantiation

Instantiations, or instances, of the class Automobile

**Object Name:** patsCar
- amount of fuel: 10 gallons
- speed: 55 miles per hour
- license plate: “135 XJK”

**Object Name:** suesCar
- amount of fuel: 14 gallons
- speed: 0 miles per hour
- license plate: “SUES CAR”

**Object Name:** ronsCar
- amount of fuel: 2 gallons
- speed: 75 miles per hour
- license plate: “351 WLF”
# UML (Universal Modeling Language)

<table>
<thead>
<tr>
<th>Class name</th>
<th>Data</th>
<th>Methods (actions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automobile</td>
<td>- fuel: double</td>
<td>- accelerate(double pedalPressure): void</td>
</tr>
<tr>
<td></td>
<td>- speed: double</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- license: String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ accelerate(double pedalPressure): void</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ decelerate(double pedalPressure): void</td>
<td></td>
</tr>
</tbody>
</table>
Objects

- **Important**: classes do not have data; individual objects have data

- Classes specify what kind of data objects have
Each Java class definition goes in its own, **SEPARATE** .java file

- ClassName → save the file as ClassName.java
- Student.java includes the class Student
What happens when you compile a .java file?
- .java file gets compiled into a .class file
  - Contains Java bytecode
  - Same filename except for .class instead of .java

You can compile a Java class before you have a program that uses it.
# class Student

<table>
<thead>
<tr>
<th>Class Name: Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Name</td>
</tr>
<tr>
<td>- Year</td>
</tr>
<tr>
<td>- GPA</td>
</tr>
<tr>
<td>- Major</td>
</tr>
<tr>
<td>- Credits</td>
</tr>
<tr>
<td>- GPA sum</td>
</tr>
<tr>
<td>+ getName</td>
</tr>
<tr>
<td>+ getMajor</td>
</tr>
<tr>
<td>+ printData</td>
</tr>
<tr>
<td>+ increaseYear</td>
</tr>
<tr>
<td>How: increase year by 1</td>
</tr>
<tr>
<td>+ calcGpa</td>
</tr>
<tr>
<td>How: average grades</td>
</tr>
</tbody>
</table>
class Student

<table>
<thead>
<tr>
<th>Class Name: Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name: String</td>
</tr>
<tr>
<td>- year: int</td>
</tr>
<tr>
<td>- gpa: double</td>
</tr>
<tr>
<td>- major: String</td>
</tr>
<tr>
<td>- credits: int</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+ getName(): String</td>
</tr>
<tr>
<td>+ getMajor(): String</td>
</tr>
<tr>
<td>+ printData(): void</td>
</tr>
<tr>
<td>+ increaseYear(): void</td>
</tr>
<tr>
<td>+ calcGpa(double grade): void</td>
</tr>
</tbody>
</table>
public class Student {
    public String name;
    public int classYear;
    public double GPA;
    public String major;
    // ...

    public String getMajor() {
        return major;
    }

    public void increaseYear() {
        classYear++;
    }
}
Create an object jack of class Student
Student jack = new Student();

Scanner keyboard = new Scanner(System.in);
Create an object keyboard of class Scanner

Return memory address of object
Assign memory address of object to variable
Create an object
Data defined in the class are called instance variables

public String name;
public int classYear;
public double GPA;
public String major;

public: no restrictions on how these instance variables are used (more details later – public is actually a bad idea here)

variables
type: int, double, String...
public class Student
{
    public String name;
    public int classYear;
    public double GPA;
    public String major;
    // ...

    public String getMajor()
    {
        return major;
    }

    public void increaseYear()
    {
        classYear++;
    }
}
Using **public** instance variables outside a class

```java
public static void main(String[] args) {
    Student jack = new Student();
    jack.name = "Jack Smith";
    jack.major = "Computer Science"

    System.out.println(jack.name + " is majoring in " + jack.major);

    Student apu = new Student();
    apu.name = "Apu Nahasapeemapetilon"
    apu.major = "Biology"

    System.out.println(apu.name + " is majoring in " + apu.major);
}
```

- `jack.name` and `apu.name` are two different instance variables because they belong to different objects
Two kinds of methods

- Methods that return a value
  - Examples: String’s .substring() method, String’s .indexOf() method, etc.

- Methods that return nothing
  - Example: System.out.println()
Methods

```java
public String getMajor()
{
    return major;
}

public void increaseYear()
{
    classYear++;  // returns nothing
}
```

- The method `getMajor()` returns a `String`.
- The method `increaseYear()` returns `void`.
Method heading: keywords
- **public**: no restriction on how to use the method (more details later)
- **void**: the method returns nothing

Method body: statements executed when the method is called (invoked)
- **Must** be inside a pair of braces

```java
public void increaseYear()
{
    classYear++;
}
```
Method printData

As usual, inside a block (defined by braces), you can have multiple statements

```java
public void printData()
{
    System.out.println("Name: "+ name);
    System.out.println("Major: "+ major);
    System.out.println("GPA: "+ gpa);
}
```
Calling methods that return nothing

- Syntax:
  - object name
  - .
  - method name
  - () -- with arguments inside, if any

- Use them as Java statements

```java
Student jack = new Student();
jack.classYear = 1;
jack.increaseYear();
System.out.println("Jack’s class year is "+ jack.classYear);
```

29
Defining methods that return a value

- **Method heading: keywords**
  - `public`: no restriction on how to use the method (more details later)
  - **Type**: the type of value the method returns

- **Method body: statements executed**
  - **Must** be inside a pair of braces
  - **Must** have a `return` statement

```java
public String getMajor()
{
    return major;
}
```
A method that returns a value must have at least one `return` statement. Terminates the method, and returns a value to the caller.

Syntax:
- `return Expression;`

`Expression` can be any expression that produces a value of type specified by the return type in the method heading.
public String getClassYear()
{
    if (classYear == 1)
        return “Freshman”;
    else if (classYear == 2)
        return “Sophomore”;
    else if ...
}
Calling methods that return a value

- object, followed by dot, then method name, then () (same as before)
- Use them as a value of the type specified by the method’s return type

Student jack = new Student();
jack.major = "Computer Science";

String m = jack.getMajor();

System.out.println("Jack’s full name is " + jack.getName());
System.out.println("Jack’s major is " + m);
Can also be used in methods that return nothing
Terminates the method
Syntax:
  - `return;`

```java
public void increaseYear()
{
    if (classYear >= 4)
        return;
    classYear++; // Here is a mistake, missing a semicolon
}
```
Monday

- More about classes
- Program 2 due