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 Instance Variables

```java
Public Class Student {
    public String name;
    public Student friend;
}

Using the Student class:
Student s1 = new Student();
s1.name = "Alan";
...
Student s2 = new Student();
s2.name = "Anna";
...
s1.friend = s2;
System.out.println( s1.name + "’s friend is " + s1.friend.name );
```
Methods

• Two kinds of methods
  – Methods that return a value
    • Examples:
      – String’s .substring() method,
      – String’s .charAt() method, etc.
  – Methods that return nothing
    • Example: System.out.println()
Methods

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

- **Returns a String**

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

- **Return type**
- **Returns nothing**
Defining methods that return nothing

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

- **Method heading: keywords**
  - `public`: no restriction on how to use the method (more details later)
  - `void`: the method returns nothing
  - Method name
  - Method parameters. No parameters in the above example

- **Method body: statements executed when the method is called (invoked)**
  - **Must be inside a pair of curly brackets**
A simple method with parameter

```java
public void increaseYear(int increment) {
    classYear += increment;
}
```

```java
public void increaseYear(int increment, boolean check) {
    if (check && classYear + increment <= MaxYear) {
        classYear += increment;
    }
}
```

Multiple parameters are separated by comma
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

• object, followed by dot, then method name, then ()
• Use them as Java statements

Student anna = new Student();
anna.classYear = 1;

anna.increaseYear();

System.out.println("Anna’s class year is "+ anna.classYear);
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 name & parameters

• Method body: statements executed
  – Must be inside a pair of braces
  – Must have a **return** statement on each execution path

```java
public String getMajor() {
    return major;
}
```
Defining methods that return a value

• Must have a **return** statement on each execution path

```java
public Color getColorFromString( String input ) {
    if (input.equals("white")) {
        return Color.white;
    } else if (input.equals("pink")) {
        return Color.pink;
    }
}
```
Defining methods that return a value

• **Must have a `return` statement on each execution path**

```java
public Color getColorFromString( String input ) {
    if (input.equals("white")) {
        return Color.white;
    } else if (input.equals("pink")) {
        return Color.pink;
    } else {
        return Color.black;
    }
}
```
return statement

• A method that returns a value must have *at least one return* statement (at least one execution path)

• **return 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
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

```java
Student allan = new Student();
allan.major = "Biostat";

String m = allan.getMajor();

System.out.println("Allan’s full name is " + allan.getName());
System.out.println("Allan’s major is " + m);
```
**return** statement

- Can also be used in methods that return nothing
- Terminates the method
- Syntax:
  ```java
  – return;
  ```

```java
public void increaseYear()
{
    if (classYear >= 4)
        return;
    classYear++;  
}
```
Code block

• A section of code enclosed by { ... }
• For grouping purpose

```java
if ( x < 0 ) {
    isPositive = false;
    x = -x;
}

for(int i = 0; i<10; i++) {
    System.out.println("*");
}
```
Code block

• Code blocks can be nested

```java
public class Hello {
    public static void main(String arg[]) {
        System.out.println("Hello.");
    }
}
```

```java
public class Hello{
    public static void main(String arg[]) {
        System.out.println("Hello.");
    }
}
```
for(int i = 0; i<100; i++) {
    if ( i % 2 ==0 ) {
        System.out.println(i + " is even");
    }
}

for(int i = 0; i<100; i++) {
    if ( i % 2 ==0 ) {
        System.out.println(i + " is even");
    }
}
Variable Scope

• The scope of a variable is the part of the program over which the variable name can be referenced.
• Variables here include instance variables (of Class) and method parameters

• Two rules:
  – You cannot refer to a variable before its declaration.
  – Variables defined in a block are only accessible from within the block
Variable Scope

• Rule 1: (Use after definition)

```java
s.nextInt();
.....
Scanner s = new Scanner(System.in);
.....
i = 6;
.....
int i;
```
Variable Scope

• Rule 1: (Use after definition)
  – Method parameters (ready for use in the method body)

```java
public Color getColorFromString(String input) {
    // input is available for use in the whole method body
    ...
}
```
Variable Scope

• Rule 2: Variables defined in a block are only accessible from within the block

```java
int outer = 1;

{
    int inner = 2;
    System.out.println("inner = "+inner);
    System.out.println("outer = "+outer);
}

System.out.println("inner = "+inner);
System.out.println("outer = "+outer);
```

Cannot reference inner here here
Variable Scope

• Rule 2: Variables defined in a block are only accessible from within the block

```java
int outer = 1;

{
    int inner = 2;
    System.out.println("inner = " + inner);
    System.out.println("outer = " + outer);
}

int inner = 3;  // why I can define inner again?
System.out.println("inner = " + inner);
System.out.println("outer = " + outer);
```
Variable Scope

• What is the scope of instance variables?
Next Class

- Continue on Class
- Define methods