COMP 110-003
Introduction to Programming
More Classes, Information Hiding and Encapsulation

February 26, 2013

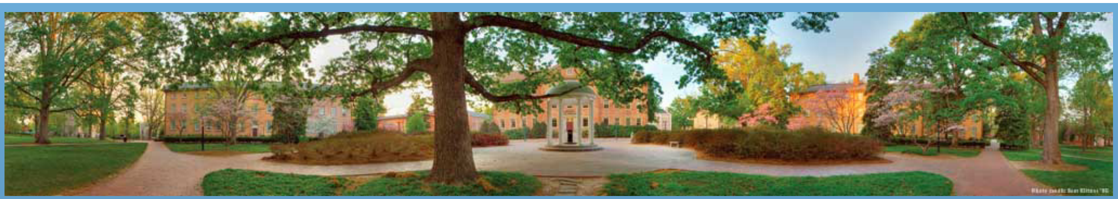
Photo credit: Sam Kittner '85

Haohan Li
TR 11:00 – 12:15, SN 011
Spring 2013

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

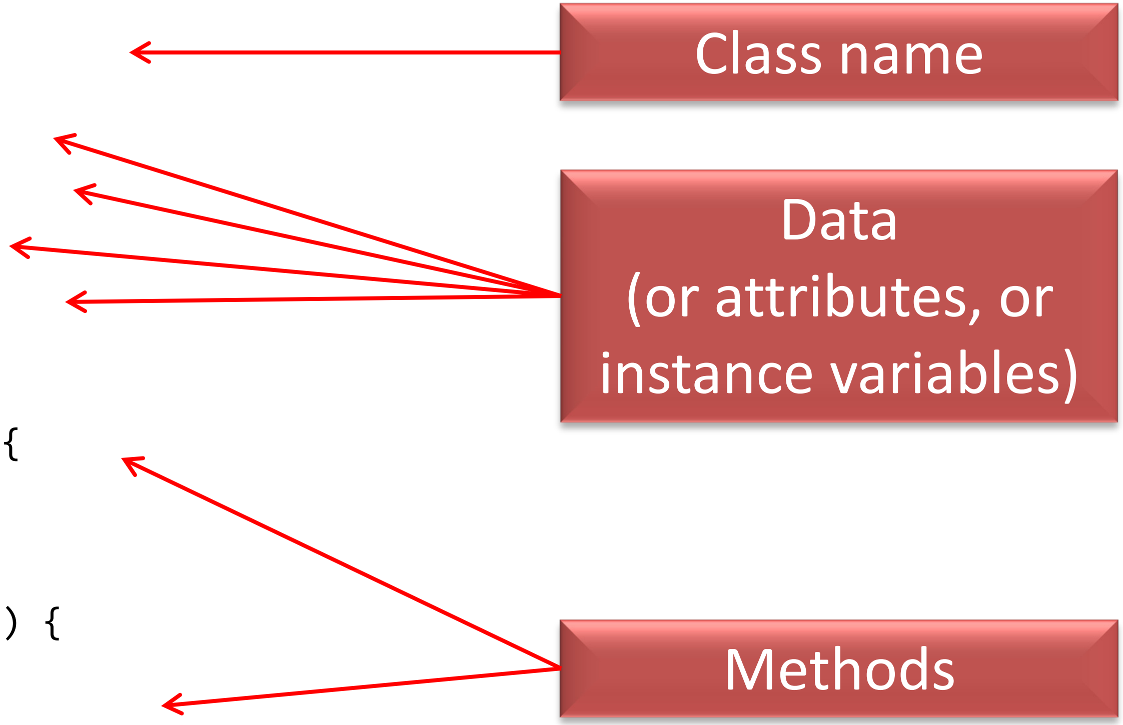
Class

- Class: a definition of a kind of object
- Object: an instance of a class
  - Contains instance variables (data) and methods
- Methods
  - Methods that return a value
  - Methods that return nothing



Defining a Class

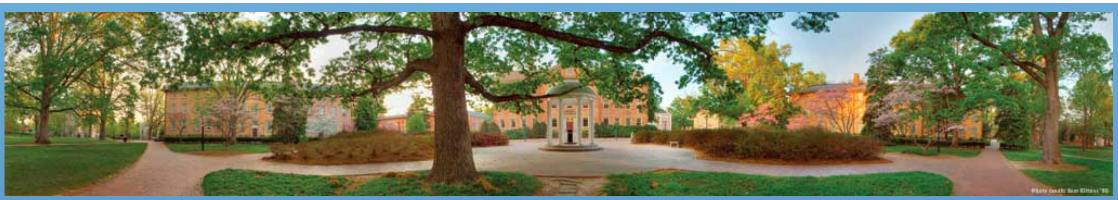
```
public class Student {
     public String name;
     public int classYear;
     public double GPA;
     public String major;

// ...

     public String getMajor() {
          return major;
     }

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



Class name

Data
(or attributes, or
instance variables)

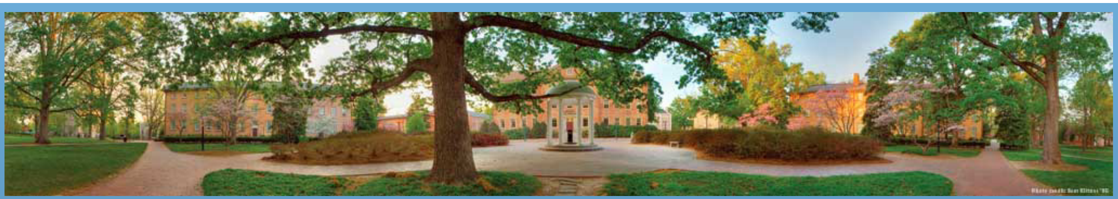
Methods



Using a Class

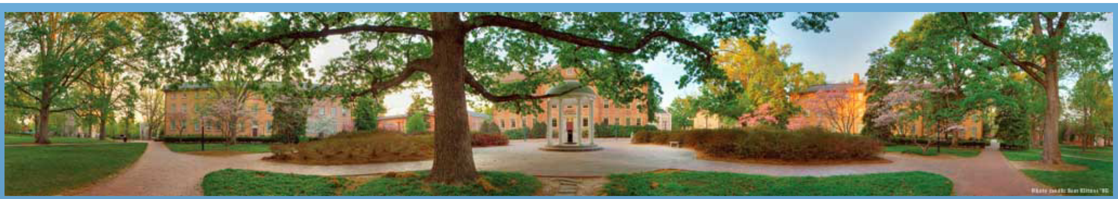
```
public class Student {
     public String name;
     public int classYear;
     public double GPA;
     public String major;

// ...

     public String getMajor() {
          return major;
     }

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

```
public class StudentTest {
  public static void main(String[] args) {
     Student jack = new Student();
     jack.name = "Jack Smith";
     jack.major = "Computer Science";
     jack.classYear = 1;
     jack.GPA = 3.5;

     String m = jack.getMajor(); //
     System.out.println("Jack's major is " + m);

     jack.increaseYear();

     System.out.println("Jack's class year is now
" + jack.classYear);

  }
}
```



Instance Variable and Local Variable

- Instance variables
  - Declared in a class
  - Confined to the class
    - Can be used anywhere in the class that declares the variable, including inside the class' methods
- Local variables
  - Declared in a method
  - Confined to the method
    - Can only be used inside the method that declares the variable

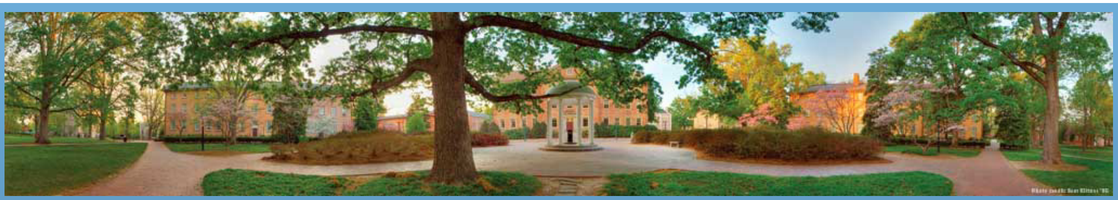


Local Variable Example

```
public class Student
{
    public String name;
    public int classYear;
    // ...

    public void printInfo()
    {
        String info = name + “: ” + classYear;
        System.out.println(info);
    }

    public void increaseYear()
    {
        classYear++;
    }

    public void decreaseYear()
    {
        classYear--;
    }
}
```

- *classYear* and *name* are instance variables
- can be used in any method in this class

- *info* is a local variable declared inside method *printInfo()*
- can only be used inside method *printInfo()*



Local Variable Example

```
public class Student
{
    public String name;
    public int classYear;
    // ...

    public void printInfo()
    {
        String info = name + “: ” + classYear;
        System.out.println(info);
    }

    public void increaseYear()
    {
        classYear++;
         info = “My info string”;  // ERROR!!!
    }

    public void decreaseYear()
    {
        classYear--;
    }
}
```

The compiler will not recognize the variable *info* inside of method *increaseYear()*

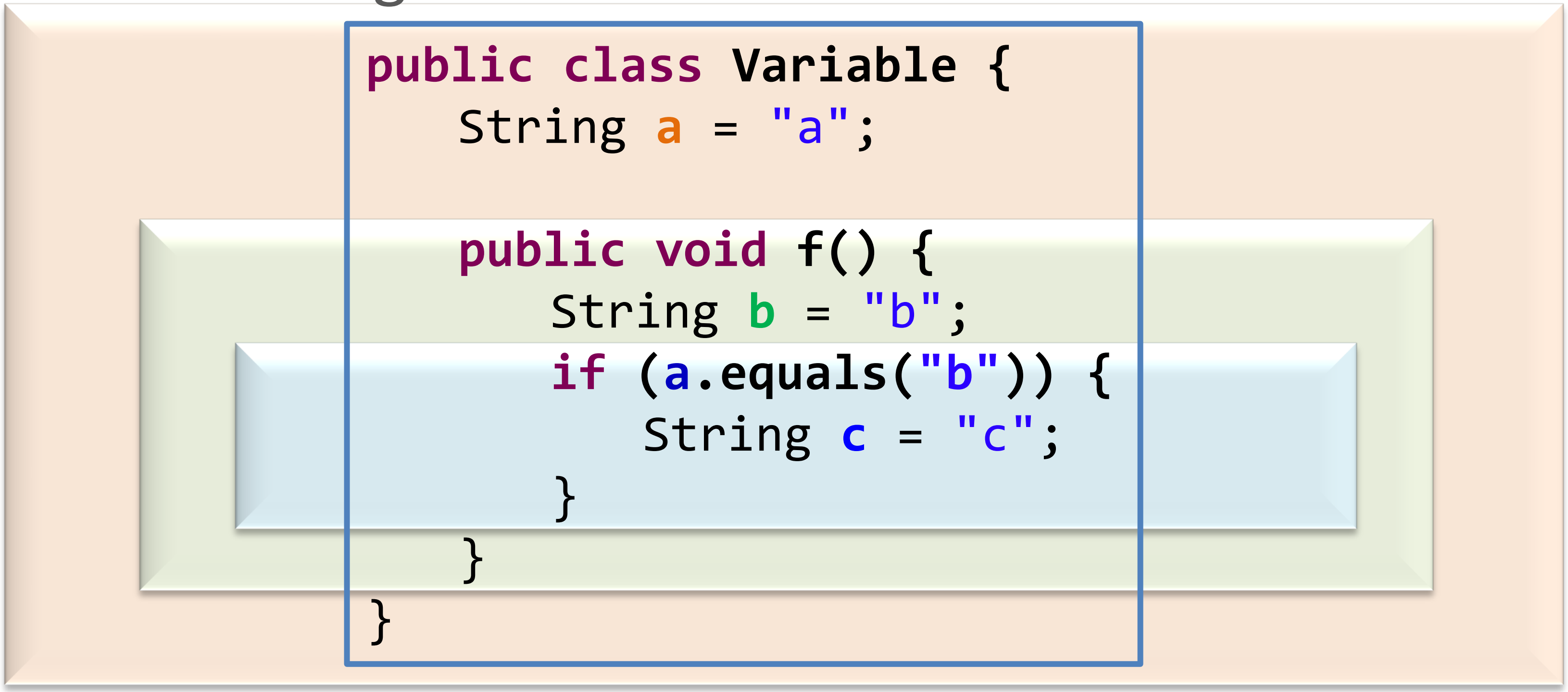
Local Variable Example

```
public class Student
{
    public String name;
    public int classYear;
    // ...

    public void printInfo()
    {
        String info = name + “: ” + classYear;
        System.out.println(info);
    }

    public void increaseYear()
    {
        classYear++;
         String info = “My info string”;  // OK
    }

    public void decreaseYear()
    {
        classYear--;
    }
}
```

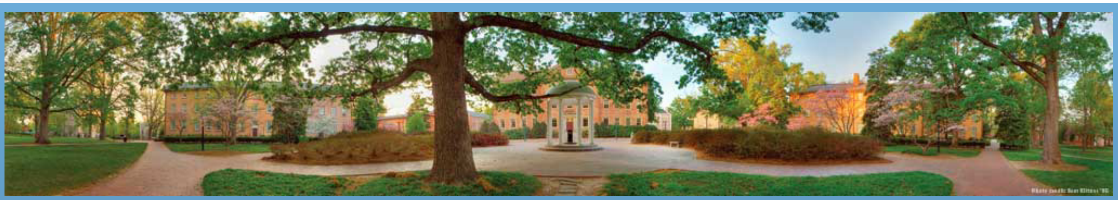
Variable *info* in *increaseYear* method not affected by variable *info* in *printInfo* method in class *Student*

Local Variable Rule

- Usually, a variable is only accessible in its surrounding brackets

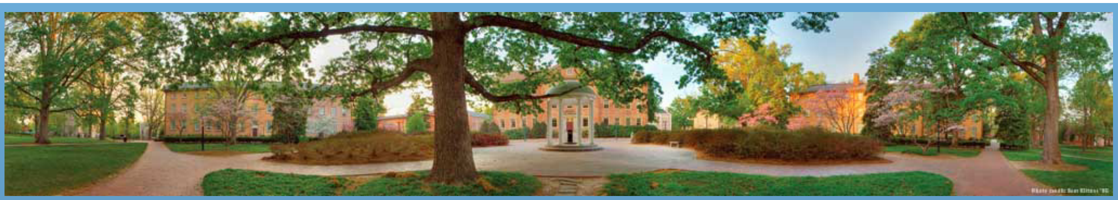


```
public class Variable {
    String a = "a";

    public void f() {
        String b = "b";
        if (a.equals("b")) {
            String c = "c";
        }
    }
}
```



Methods with Parameters

- Compute the square of this number
  - 5
  - 10
  - 7
- I could give you any number, and you could tell me the square of it
- We can do the same thing with methods



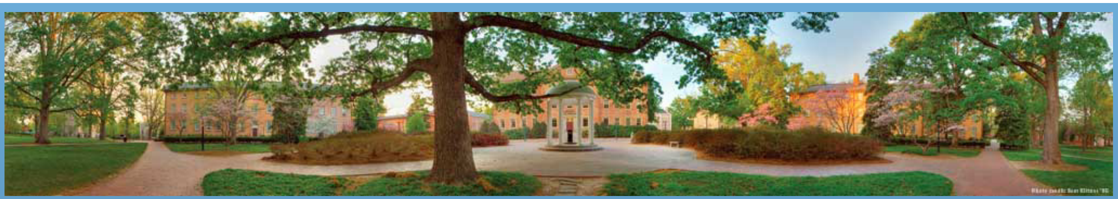
Methods with Parameters

- Parameters are used to hold the value that you pass to the method

- Parameters can be used as (local) variables inside the method

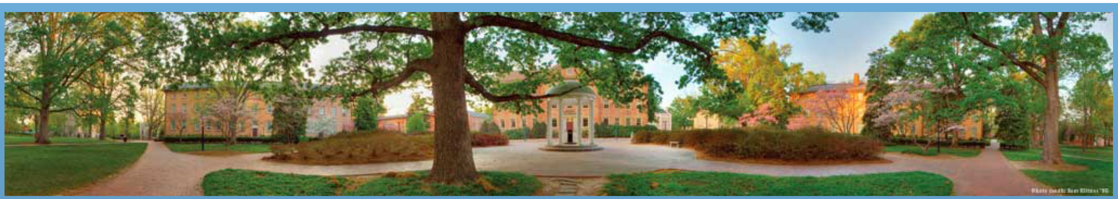
```
public int square(int number)
{
    return number * number;
}
```

Parameters go inside the parentheses of method header



Calling a Method with Parameters

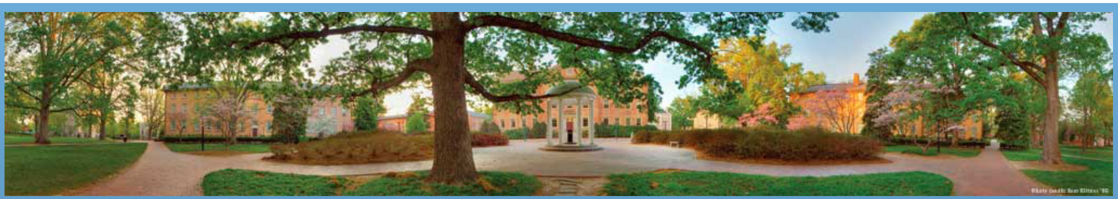
```
public class Student
{
    public String name;
    public int classYear;
    // …
    public void setName(String studentName)
    {
        name = studentName;
    }
    public void setClassYear(int year)
    {
        classYear = year;
    }
}
```



Calling a Method with Parameters

```
public static void main(String[] args)
{
    Student jack = new Student();
    jack.setName(“Jack Smith”);
    jack.setClassYear(3);
}
```

Parameters/
Arguments

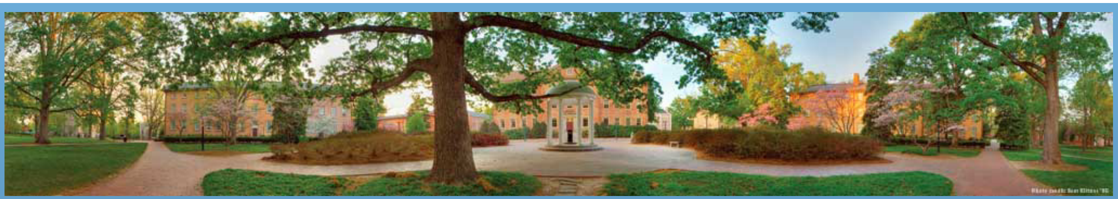


Methods with Multiple Parameters

- Multiple parameters separated by commas

```
public double getTotal(double price, double tax)
{
  return price + price * tax;
}
```

- When calling a method, the order, type, and number of arguments must match parameters specified in method heading



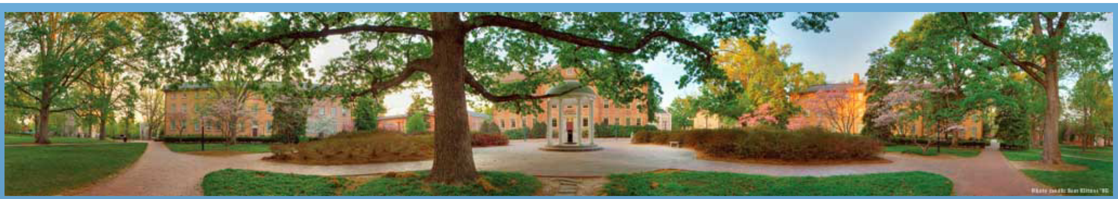
Methods with Multiple Parameters

```
public class SalesComputer
{
    public double getTotal(double price, double tax)
    {
        return price + price * tax;
    }
// …
SalesComputer sc = new SalesComputer();
~~double total = sc.getTotal(“19.99”, Color.RED);~~
~~double total = sc.getTotal(19.99);~~
double total = sc.getTotal(19.99, 0.065);
int price = 50;
total = sc.getTotal(price, 0.065);
```

Automatic typecasting

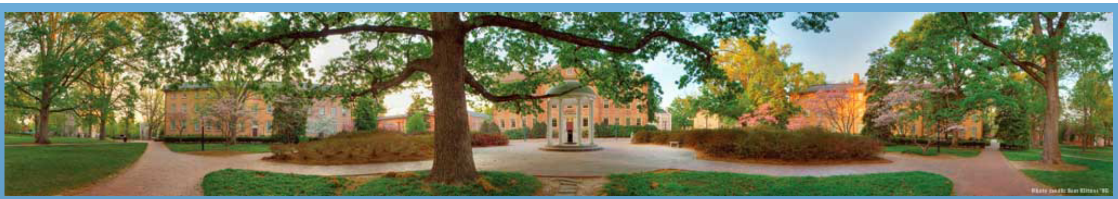
Calling Methods from Methods

- A method body can call another method
  - Done the same way:
    `receiving_object.method();`
- If calling a method in the same class, do not need receiving_object:
  - `method();`
- Alternatively, use the this keyword (can be omitted)
  - `this.method();`



Calling Methods from Methods

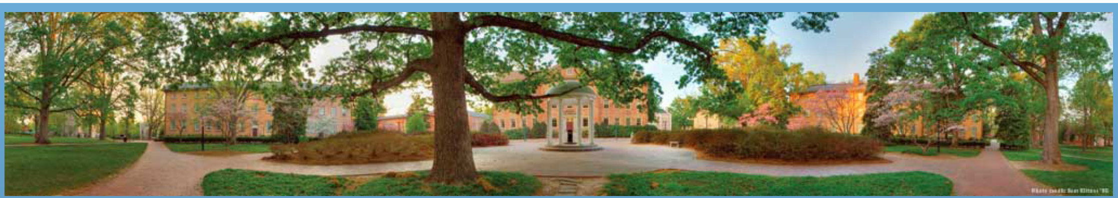
```
public class Student
{
    public String name;
    public int classYear;
    public void setName(String studentName)
    {
        name = studentName;
    }
    public void setClassYear(int year)
    {
        classYear = year;
    }
    public void setNameAndYear(String studentName, int year){
        this.name = studentName; // or this.setName(studentName);
        this.classYear = year; // or this.setClassYear(year);
    }
}
```



public/private Modifier

- public void setMajor()
- public int classYear;

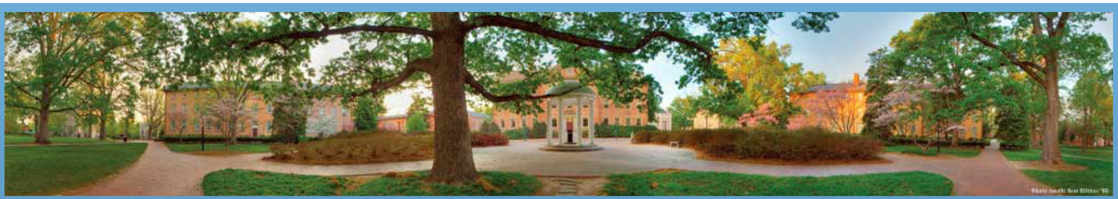
- public: there is no restriction on how you can use the method or instance variable



public/private Modifier

- private void setMajor()
- private int classYear;

- private: can not directly use the method or instance variable’s name outside the class

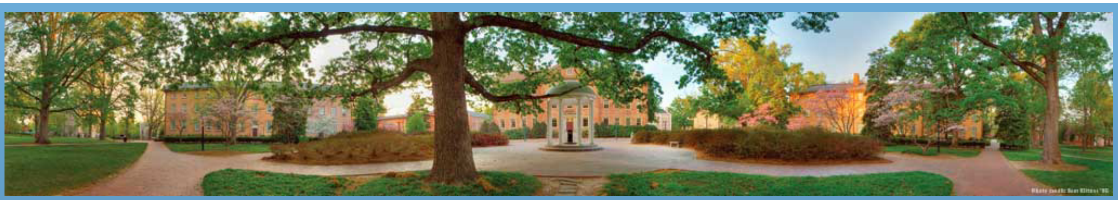


public/private Modifier

```
public class Student
{
     public int classYear;
     private String major;
}
public class StudentTest{
   public static void main(String[] args){
        Student jack = new Student();
        jack.classYear = 1;
        jack.major = “Computer Science”; // ERROR!!!
   }
}
```

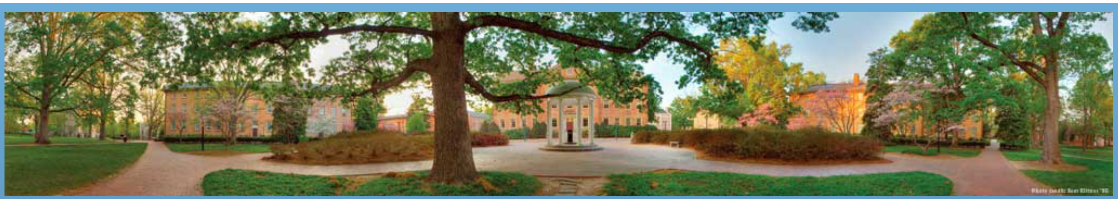
OK, *classYear* is *public*

Error!!! *major* is *private*



More about private

- Hides instance variables and methods inside the class/object. The private variables and methods are still there, holding data for the object.
- Invisible to external users of the class
  - Users cannot access private class members directly
- **Information hiding**



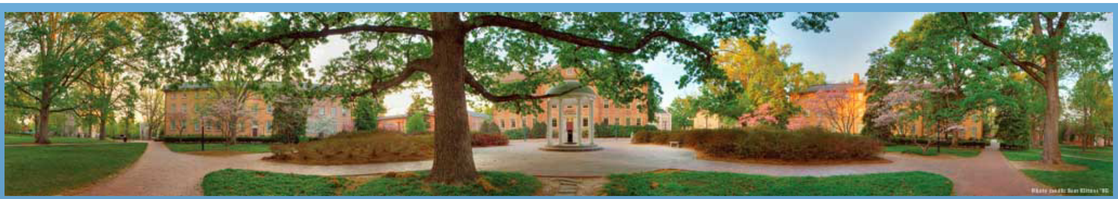
Example: Rectangle

```
public class Rectangle
{
    public int width;
    public int height;
    public int area;

    public void setDimensions(
        int newWidth,
        int newHeight){
        width = newWidth;
        height = newHeight;
        area = width * height;
    }

    public int getArea(){
        return area;
    }
}
```

```
Rectangle box = new Rectangle();
box.setDimensions(10, 5);
System.out.println(box.getArea());


// Output: 50


box.width = 6;
System.out.println(“The rectangle
   with edges ” + box.width + “
   and ” + box.height + “ has area
   size ” + box.getArea());


// Output: The rectangle with
   edges 6 and 5 has area size 50

// Wrong answer!
```

Accessors and Mutators

- How do you access private instance variables?
- Accessor methods (a.k.a. get methods, **getters**)
  - Allow you to look at data in private instance variables
- Mutator methods (a.k.a. set methods, **setters**)
  - Allow you to change data in private instance variables

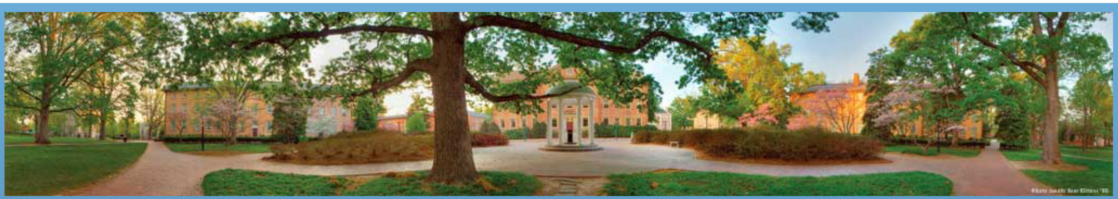


Example: Student

```
public class Student
{
    private String name;
    private int age;

    public void setName(String studentName) {
        name = studentName;
    }
    public void setAge(int studentAge) {
        age = studentAge;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}
```

Mutators

Accessors



Example: Student

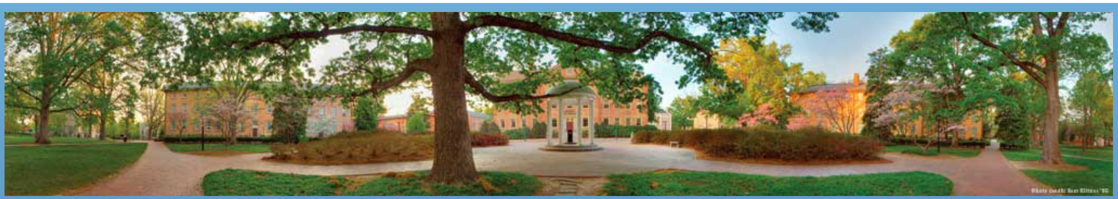
```
public class Student
{
    private String name;
    private int age;

    public void setName(String studentName) {
        name = studentName;
    }
    public void setAge(int studentAge) {
      if (studentAge > 0)
        age = studentAge;
      else System.out.println(“The input for age should be positive”)
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}
```

Mutators

Accessors

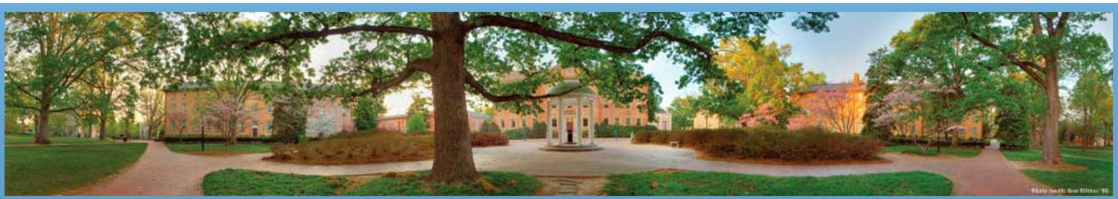
Private Methods

- **Why make methods private?**
- Helper methods that will only be used from inside a class should be private
  - External users have no need to call these methods
- **Encapsulation**



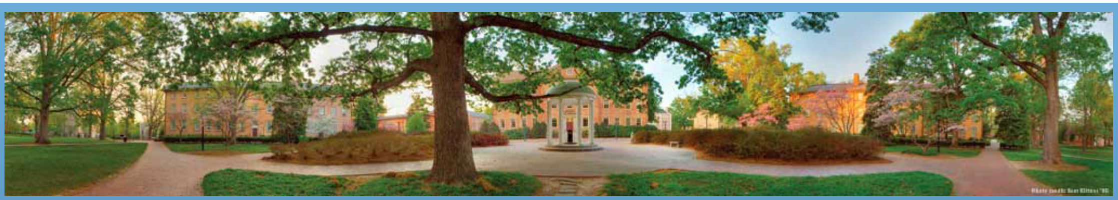
Private Methods

```java
public class RightTriangle {
    private double side_a;
    private double side_b;

    private double square(double d) {
        // some calculation
    } // don’t want others to use – rounded for rounded output

    private double sqrt(double d) {
        // some complicated calculation
    } // don’t want others to use – optimized for triangle only

    public double getSideC() {
        return this.sqrt(this.square(side_a) + this.square(side_b));
    }
}
```



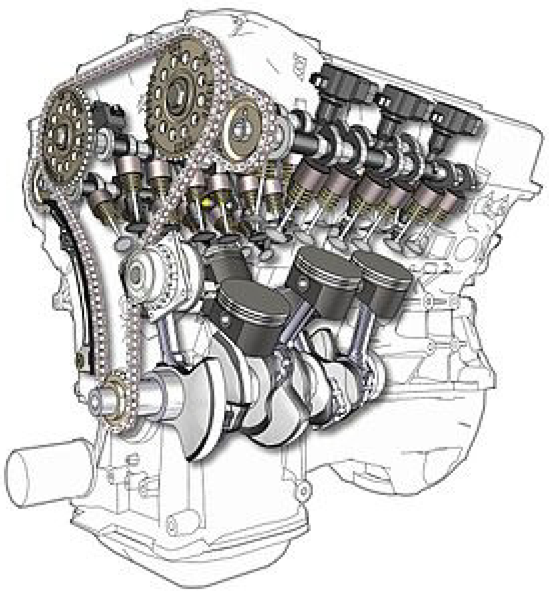
Example: Driving a Car

- Accelerate with the accelerator pedal
- Decelerate with the brake pedal
- Steer with the steering wheel
- Does not matter if:
  - You have a 4-cylinder engine or a 6-cylinder engine
  - Especially, you don’t have to control how many valves shall be on at each second in order to drive a car
- You still drive the same way



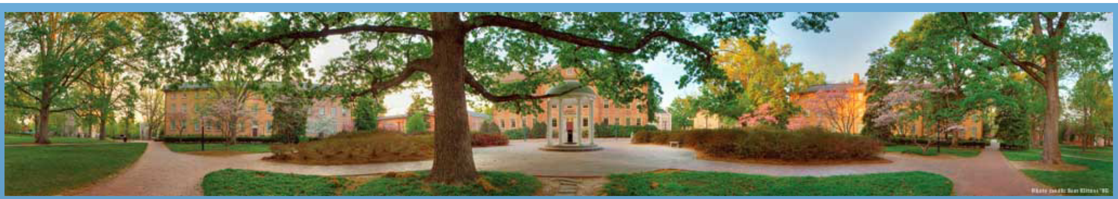
Encapsulation

- The *__interface__* is the same
- The underlying *__implementation__* may be different



Encapsulation in Classes

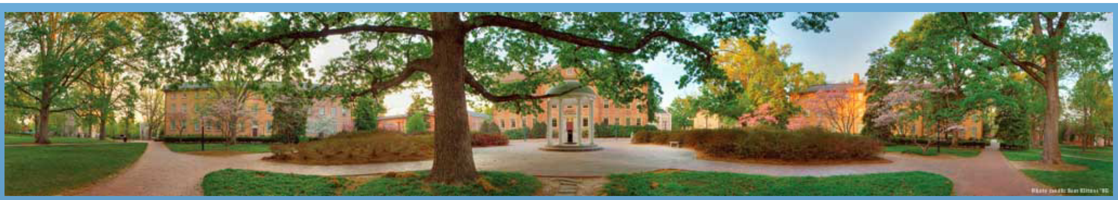
- A *class interface* tells programmers all they need to know to use the class in a program
- The *implementation* of a class consists of the private elements of the class definition
  - private instance variables and constants
  - private methods
  - bodies of public methods



Example: Two Rectangle Classes

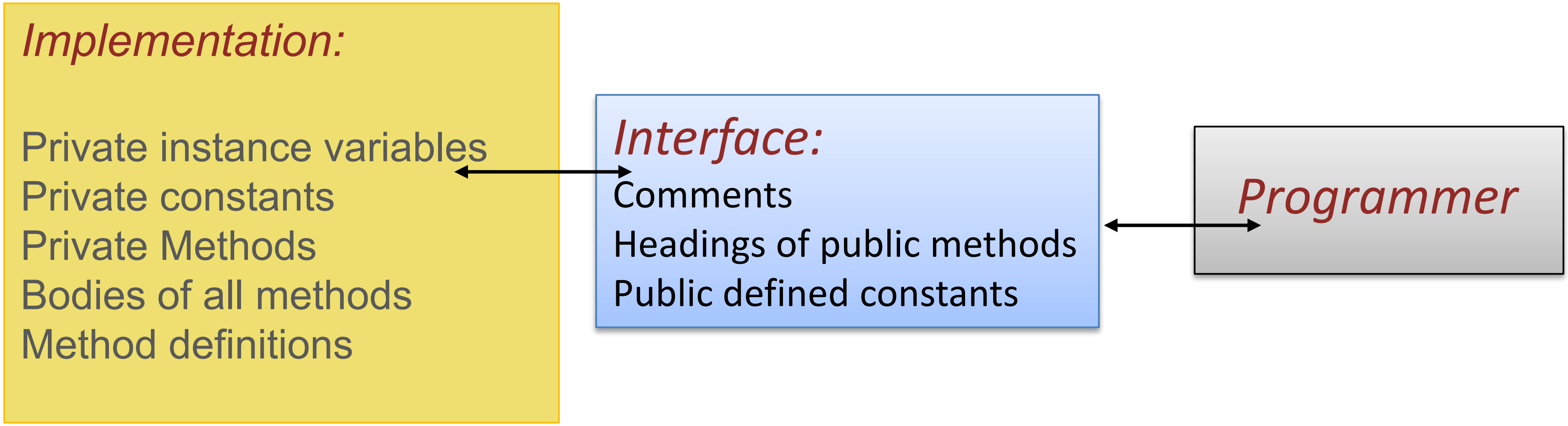
```
public class Rectangle
{
    private int width;
    private int height;
    private int area;

    public void setDimensions(
        int newWidth,
        int newHeight)
    {
        width = newWidth;
        height = newHeight;
        area = width * height;
    }

    public int getArea()
    {
        return area;
    }
}
```

```
public class Rectangle
{
    private int width;
    private int height;

    public void setDimensions(
        int newWidth,
        int newHeight)
    {
        width = newWidth;
        height = newHeight;
    }

    public int getArea()
    {
        return width * height;
    }
}
```



Well Encapsulation

- Imagine a wall between (other) programmers and (your) implementation
  - It’s called interface



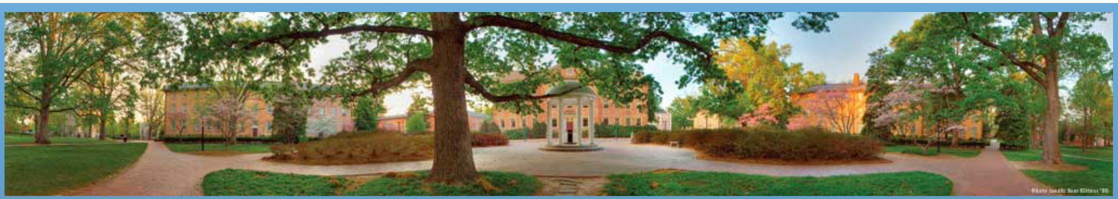
Implementation:

Private instance variables
Private constants
Private Methods
Bodies of all methods
Method definitions

Interface:

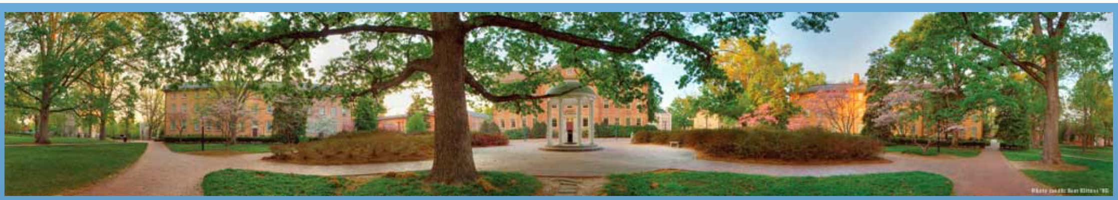
Comments
Headings of public methods
Public defined constants

Programmer



Guidelines When You Define a Class

- Comments before class definition (this is your header)
- Instance variables are *private*
- Provide *public* accessor and mutator methods
- Comments before methods
- Make helping methods *private*
- /* */ for user-interface comments and // for implementation comments



Initialization of Instance Variables

- You can declare default values for instance variables

```
public class Rectangle
{
    public int width = 1;
    public int height = 1;
    public int area = 1;
    public void setDimensions(
        int newWidth,
        int newHeight){
        width = newWidth;
        height = newHeight;
        area = width * height;
    }
    public int getArea(){
        return area;
    }
}
```

```
Rectangle box = new Rectangle();
System.out.println(box.getArea());

// Output: 1
```

Select Proper Instance Variables

```
public class Rectangle
{
    private int width;
    private int height;
    private int area;
    public void setDimensions(
        int newWidth,
        int newHeight){
        width = newWidth;
        height = newHeight;
        area = width * height;
    }
    public void setWidth(
        int newWidth){
        width = newWidth;
        area = width * height;
    }
    public void setHeight(
        int newHeight){
        height = newHeight;
        area = width * height;
    }
    public int getArea(){
        return area;
    }
}
```

```
public class Rectangle
{
    private int width;
    private int height;
    public void setDimensions(
        int newWidth,
        int newHeight){
        width = newWidth;
        height = newHeight;
      }
      public void setWidth(
        int newWidth){
        width = newWidth;
      }
      public void setHeight(
        int newHeight){
        height = newHeight;
      }
    public int getArea(){
        return width * height;
      // MUCH SHORTER AND LESS
      // POSSIBILITY OF MAKING MISTAKES
      }
}
```