THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

# COMP 110
# Introduction to Programming

**Thursday October 23, 2014**

Jay Aikat
Fall 2014
TR 9:30 – 10:45, GS-G100

## Previous Class

- What did we discuss?

# Announcements

- **Midterm – grades released**
- **Midterms returned today**
- **LAB5 – Due Mon, Oct 27**
- **Assignment4 – Due Thu, Oct 30**



# Using more than one 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);

  }
}
```

## Control Flow

- Program control flow
  - execution always begins with the first statement in the method `main`
  - other methods execute only when called
- Method control flow
  - when a method is invoked, the ***flow of control*** jumps to the method and the computer executes its code
  - when complete, the flow of control returns to the place where the method was called and the computer continues executing code



## 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



# 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

# Constructors

- Create and initialize new objects
- Special methods that are called when (and **only** when) creating a new object

```
Student jack = new Student();
```

Calling a constructor



# Creating an Object

Create an object *jack* of class *Student*

```
Student jack = new Student();
```

Assign the memory address of the object to variable

Return memory address of object

Create an object by calling a constructor

```
Scanner keyboard = new Scanner(System.in);
```

Create an object *keyboard* of class *Scanner*

## Constructors

- Can perform any action you write into a constructor's definition
  - There are no specific rules about what's in a constructor
- Meant to perform initializing actions
  - Usually, initializing values of instance variables by the creator of the object



## Similar to Setter Methods

- However, constructors *create* an object in addition to setting the values of instance variables
- Like methods, constructors can have parameters

# Example: Pet Class

```
public class Pet
{
    private String name;
    private int age;
    private double weight;

    public Pet()
    {
        name = "No name yet.";
        age = 0;
        weight = 0;
    }

    public static void main(String[] args)
    {
                Pet p = new Pet();
        }
}
```

Default constructor

Call constructor



# The Same as Initialization

```
public class Pet
{
    private String name = "No name yet.";
    private int age = 0;
    private double weight = 0;


    public static void main(String[] args)
    {
                Pet p = new Pet();
        }
}
```

Default constructor not declared – but still exists

Call default constructor (so an object is created)

## Default Constructor

- Constructor that takes no parameters

```
public Pet()
{
    name = "No name yet.";
    age = 0;
    weight = 0;
}
```

- Java automatically defines a default constructor if you do not define any constructors
  - You've never written a constructor but you can still create objects



## Next class (Tue, Oct 28)

- More on methods…