

# AP Computer Science B

## Java Object-Oriented Programming [Ver. 2.0]

### Unit 4: Object-Oriented Design

WEEK 2: CHAPTER 10 CLASSES AND OBJECTS (PART 2: DATA ORGANIZATION)

DR. ERIC CHOU

IEEE SENIOR MEMBER



# Objectives

---

- Class Classification: Data, Utility, Wrapper, Functional, Tester, and Handler
- Data Classes
- Classes Testes in AP Computer Science A Exam
- Data Containers: Objects as Containers, Array of Objects, and Object of Arrays
- Scope and Visibility of Member Fields and Functions
- Building a Package



# Classes

Data, Utility, Wrapper,  
Functional, Tester, and  
Handler

---

LECTURE 1



# Different Usages of Classes

---

- (1) Main Application Class: Class with a public static void main() function
- (2) Test/Demo Class: Class for testing other class (or classes)
- (3) Library Function (Utility) Class: Class provides static methods for other program or classes to use as a library function. Example class: Math Class, java.util.Arrays Class. This can also be user-defined.
- (4) Data Class: Class used as a collection of data such as a record.
- (5) Program Class: Class used as a collection of programs such a module.
- (6) Helper Class: used to assist in providing some functionality, which isn't the main goal of the application or class in which it is used. (Delegation)
- (7) GUI component Class: Classes directly mapped to a GUI component.
- (8) Other API Classes .



# [1] Main Application Class

---

Every Java application must contain a main method whose signature looks like this:

```
public static void main(String[] args)
```

The method signature for the main method contains three modifiers:

- **public** indicates that the main method can be called by any object. Controlling Access to Members of a Class(in the Writing Java Programs trail) covers the ins and outs of the access modifiers supported by the Java language.
- **static** indicates that the main method is a class method. Instance and Class Members(in the Writing Java Programs trail) talks about class methods and variables.
- **void** indicates that the main method does not return any data.



# [1] Main Application Class

---

The first bold line in the following listing begins the definition of a main method.

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
    }
}
```

doesn't return any value.



## [2] Tester Class

---

Tester Class is a special kind of Java Class which does the following things:

- (1) Prepare test patterns to test a class which is our DUT (Design Under Test).
- (2) Collect the output from the DUT and provide performance evaluation statistics.
- (3) Control over the testing flow (Iterative test, Conditional test, Monte Carlo test, and ...)



# [3] Utility Class (Library Function Class)

## Usually with Static Members

---

- In computer programming, a **utility class** is a class that defines a set of methods that perform common, often re-used functions. Most utility classes define these common methods under static (see **Static** variable) scope.
- Examples of utility classes include **java.util.Collections** which provides several utility methods (such as sorting) on objects that implement a Collection (`java.util.Collection`).
- `Math`, `java.util.Scanner`, `java.util.Arrays`, `java.io.File`, ...





# [4] Data Class

Serve as data record template. (Refers to Data Encapsulation Lecture)

---

```
Class Student {  
    private String name = "Your name";  
    private int studentID = 0;  
    private int mathScore = 0;  
    private int englishScore = 0;  
    public String getName(){ return name; }  
    public int getStudentID(){ return studentID; }  
    public int getMathScore() {return mathScore; }  
    public int getEnglishScore() {return englishScore; }  
    public void setName(String n){ name = n; }  
    public void getStudentID(int id){ studentID= id; }  
    public void getMathScore(int s) { mathScore=s; }  
    public void getEnglishScore(int s) { englishScore=s; }  
}
```

# [5] Program Class

## Using multiple classes in Java program

```
class Computer {
    Computer() {
        System.out.println("Constructor of Computer class.");
    } // Constructor as program loader
    void computer_method() {
        System.out.println
            ("Power gone! Shut down your PC soon...");
    }
    public static void main(String[] args) {
        Computer my = new Computer(); // load sub-programs
        Laptop your = new Laptop();
        Notebook his = new Notebook();
        my.computer_method();          // run sub-programs
        your.laptop_method();
        his.notebook_method();
    }
}
```

```
Class Notebook {
    notebook_top() {
        System.out.println
            ("Constructor of Notebook class.");
    } // Constructor as program loader
    void notebook_method() {
        System.out.println("99% Battery available.");
    }
}

class Laptop {
    Laptop() {
        System.out.println
            ("Constructor of Laptop class.");
    } // Constructor as program loader
    void laptop_method() {
        System.out.println("99% Battery available.");
    }
}
```



## [6] Helper class

---

- In object-oriented programming, a helper class is used to assist in providing some functionality, which isn't the main goal of the application or class in which it is used. An instance of a helper class is called a helper object (for example, in the delegation pattern).
- **Helper classes** are often created in **introductory programming lessons**, after the novice programmer has moved beyond creating one or two classes.
- A **utility class** is a special case of a helper class in which the methods are all **static**. In general, helper classes do not have to have all static methods, and may have instance variables and multiple instances of the helper class may exist.



# [7] GUI Packages

## A Collection of GUI Classes

---

A **GUI package** contains the core GUI graphics classes:

- GUI Component classes (such as Button, TextField, and Label),
- GUI Container classes (such as Frame, Panel, Dialog and Scroll Pane),
- Layout managers (such as Flow Layout, Border Layout and Grid Layout),
- Custom graphics classes (such as Graphics, Color and Font).

The **GUI event package** supports event handling:

- Event classes (such as Action Event, Mouse Event, Key Event and Window Event),
- Event Listener Interfaces (such as Action Listener, Mouse Listener, Key Listener and Window Listener),
- Event Listener Adapter classes (such as Mouse Adapter, Key Adapter, and Window Adapter).



# Data Class

---

## LECTURE 2



# Data Class

---

- A data class refers to a class that contains only **fields** and crude methods for accessing them (**getters** and **setters**).
- These are simply containers for data used by other classes.
- These classes don't contain any additional functionality and can't independently operate on the data that they own.



# Reasons for the Problem

---

- It's a normal thing when a newly created class contains only a few public fields (and maybe even a handful of getters/setters).
- But the true power of objects is that they can contain behavior types or operations on their data.



# Treatment

---

- If a class contains public fields, use **Encapsulate Field** to hide them from direct access and require that access be performed via getters and setters only.
- Use **Encapsulate Collection** for data stored in collections (such as arrays).
- Review the client code that uses the class. In it, you may find functionality that would be better located in the data class itself. If this is the case, use **Move Method** and **Extract Method** to migrate this functionality to the data class.





# Treatment

---

- After the class has been filled with well thought-out methods, you may want to get rid of old methods for data access that give overly broad access to the class data. For this, **Remove Setting Method** and **Hide Method** may be helpful.



# Data Classes from Java Library

---

LECTURE 3



# Using Classes from the Java Library

## The Date Class

---

- Java provides a system-independent encapsulation of date and time in the java.util.Date class.
- You can use the Date class to create an instance for the current date and time and use its toString method to return the date and time as a string.



# Using Classes from the Java Library

## The Date Class

---

The + sign indicates  
public modifier



java.util.Date	
+Date()	
+Date(elapseTime: long)	
+toString(): String	
+getTime(): long	
+setTime(elapseTime: long): void	

Constructs a Date object for the current time.

Constructs a Date object for a given time in  
milliseconds elapsed since January 1, 1970, GMT.

Returns a string representing the date and time.

Returns the number of milliseconds since January 1,  
1970, GMT.

Sets a new elapse time in the object.

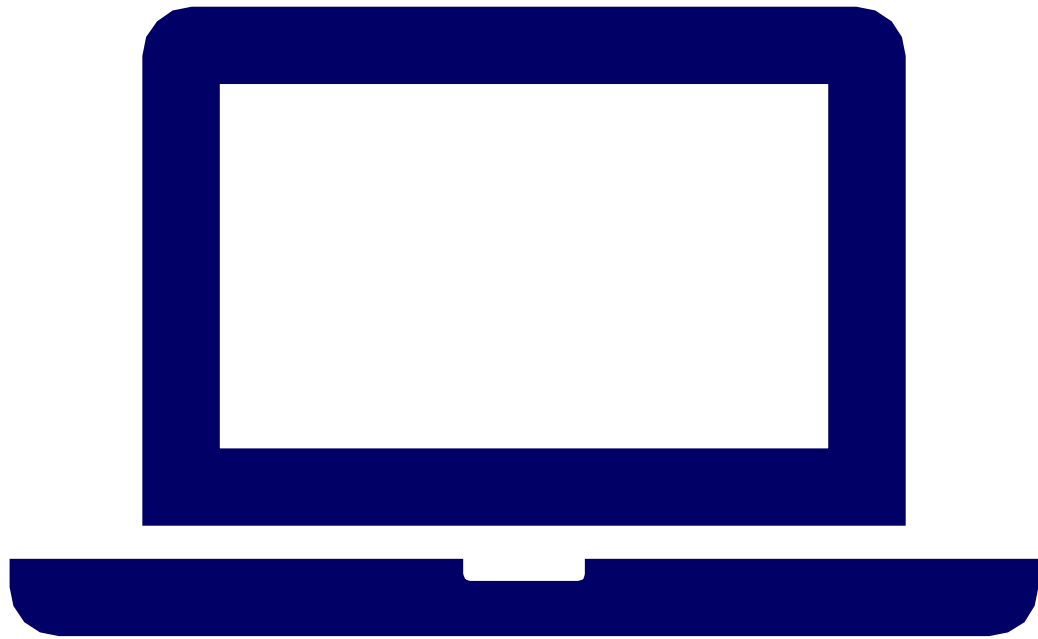
The Date Class  
Example  
DateExample.java

For example, the following code

```
java.util.Date date = new  
    java.util.Date();  
System.out.println(date.toString());
```

displays a string like Sun Mar 09  
13:50:19 EST 2003.

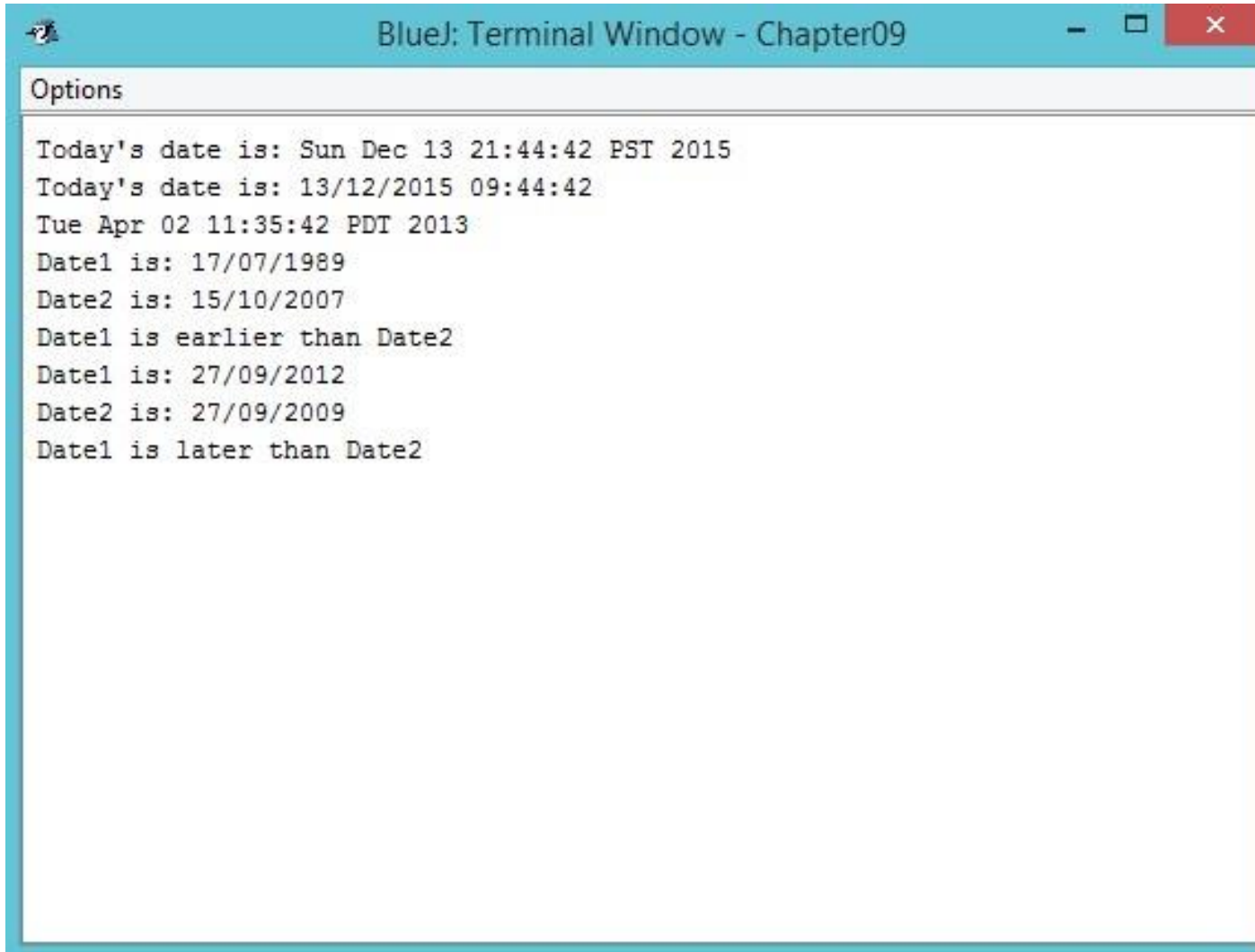
Go BlueJ.



# Demonstration Program

---

DATEEXAMPLE.JAVA

A screenshot of a BlueJ terminal window titled "BlueJ: Terminal Window - Chapter09". The window has a light blue title bar with standard window controls (minimize, maximize, close). Below the title bar is a tab labeled "Options". The main area of the window displays the output of a Java program, showing the current date and time in two different formats, and then comparing two dates.

```
Options
Today's date is: Sun Dec 13 21:44:42 PST 2015
Today's date is: 13/12/2015 09:44:42
Tue Apr 02 11:35:42 PDT 2013
Date1 is: 17/07/1989
Date2 is: 15/10/2007
Date1 is earlier than Date2
Date1 is: 27/09/2012
Date2 is: 27/09/2009
Date1 is later than Date2
```

Result:  
DateExample.java



# Calendar Class

---

Provide date  
information in  
certain calendar  
format.

Go BlueJ !!!



## *java.util.Calendar*

```
#Calendar()  
+get(field: int): int  
+set(field: int, value: int): void  
+set(year: int, month: int,  
    dayOfMonth: int): void  
+getActualMaximum(field: int): int  
+add(field: int, amount: int): void  
+getTime(): java.util.Date  
+setTime(date: java.util.Date): void
```

Constructs a default calendar.

Returns the value of the given calendar field.

Sets the given calendar to the specified value.

Sets the calendar with the specified year, month, and date. The month parameter is 0-based; that is, 0 is for January.

Returns the maximum value that the specified calendar field could have.

Adds or subtracts the specified amount of time to the given calendar field.

Returns a `Date` object representing this calendar's time value (million second offset from the UNIX epoch).

Sets this calendar's time with the given `Date` object.



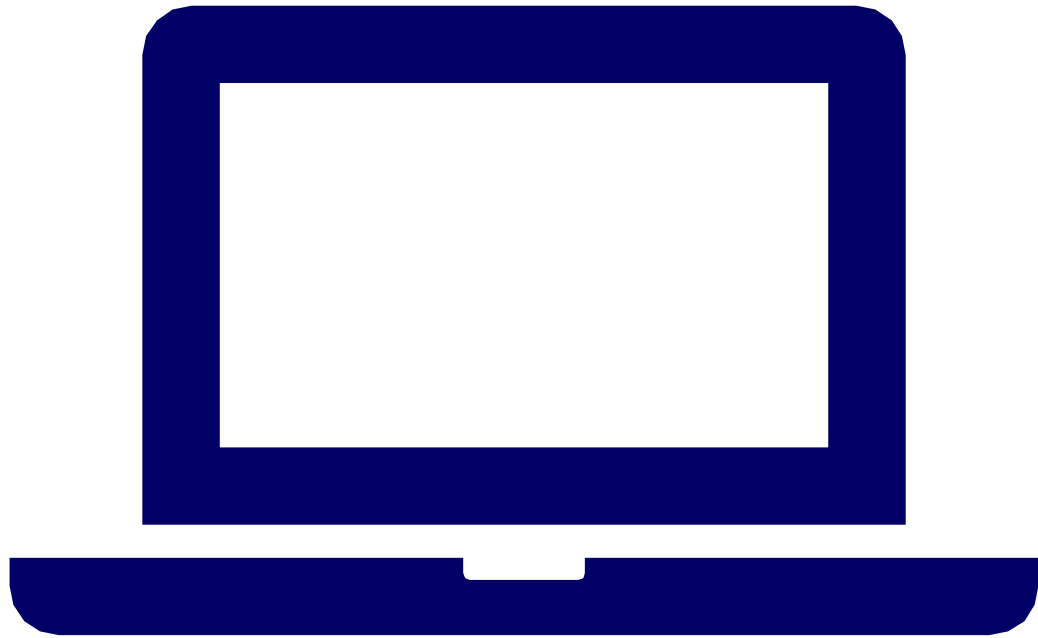
## *java.util.GregorianCalendar*

```
+GregorianCalendar()  
+GregorianCalendar(year: int,  
    month: int, dayOfMonth: int)  
+GregorianCalendar(year: int,  
    month: int, dayOfMonth: int,  
    hour: int, minute: int, second: int)
```

Constructs a `GregorianCalendar` for the current time.

Constructs a `GregorianCalendar` for the specified year, month, and date.

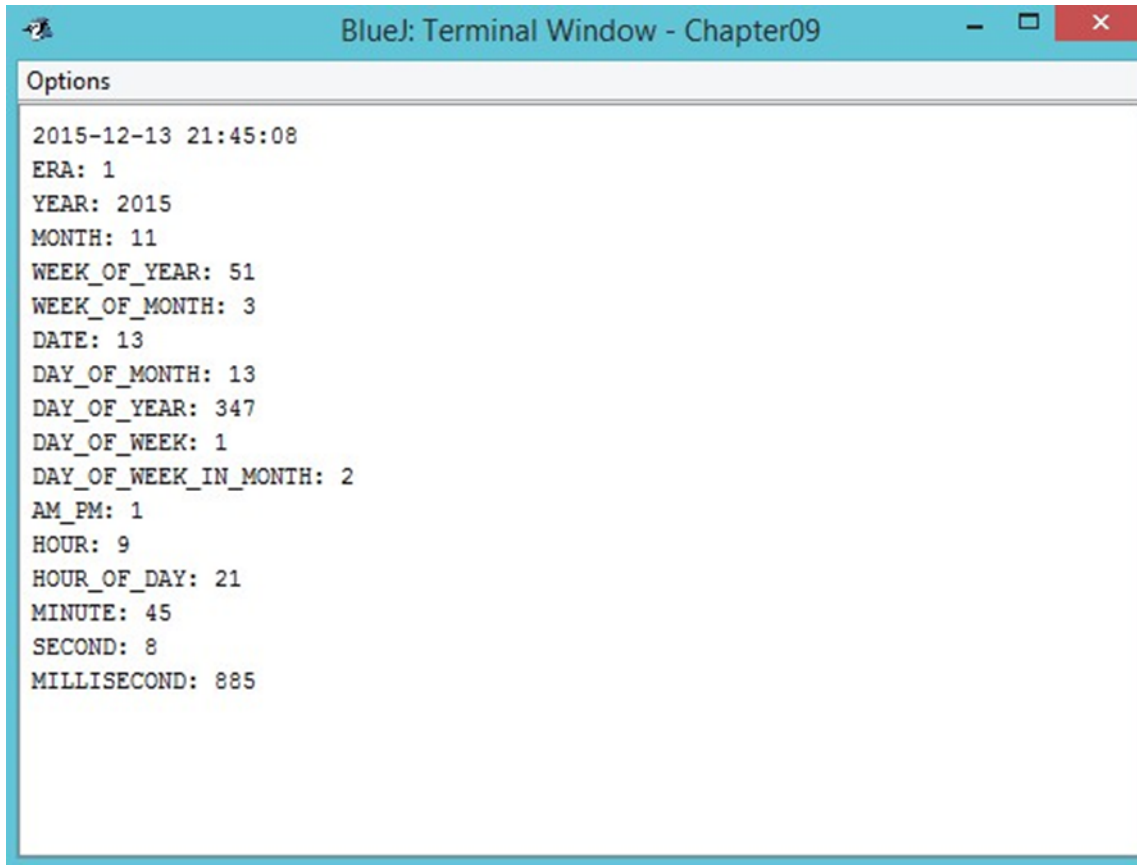
Constructs a `GregorianCalendar` for the specified year, month, date, hour, minute, and second. The month parameter is 0-based, that is, 0 is for January.



# Demonstration Program

---

CALENDAREXAMPLE.JAVA



A screenshot of a Java IDE terminal window titled "BlueJ: Terminal Window - Chapter09". The terminal displays the output of a program, starting with a timestamp "2015-12-13 21:45:08" followed by various date and time fields in uppercase letters separated by colons. The fields include ERA, YEAR, MONTH, WEEK\_OF\_YEAR, WEEK\_OF\_MONTH, DATE, DAY\_OF\_MONTH, DAY\_OF\_YEAR, DAY\_OF\_WEEK, DAY\_OF\_WEEK\_IN\_MONTH, AM\_PM, HOUR, HOUR\_OF\_DAY, MINUTE, SECOND, and MILLISECOND.

```
Options
2015-12-13 21:45:08
ERA: 1
YEAR: 2015
MONTH: 11
WEEK_OF_YEAR: 51
WEEK_OF_MONTH: 3
DATE: 13
DAY_OF_MONTH: 13
DAY_OF_YEAR: 347
DAY_OF_WEEK: 1
DAY_OF_WEEK_IN_MONTH: 2
AM_PM: 1
HOUR: 9
HOUR_OF_DAY: 21
MINUTE: 45
SECOND: 8
MILLISECOND: 885
```

Result:

---

CalendarExample.java



# Point2D Class

---

- Java API has a convenient Point2D class in the **javafx.geometry** package for representing a point in a two-dimensional plane.
- The UML diagram for the class is shown in the figure on the right.

## **javafx.geometry.Point2D**

+Point2D(x: double, y: double)

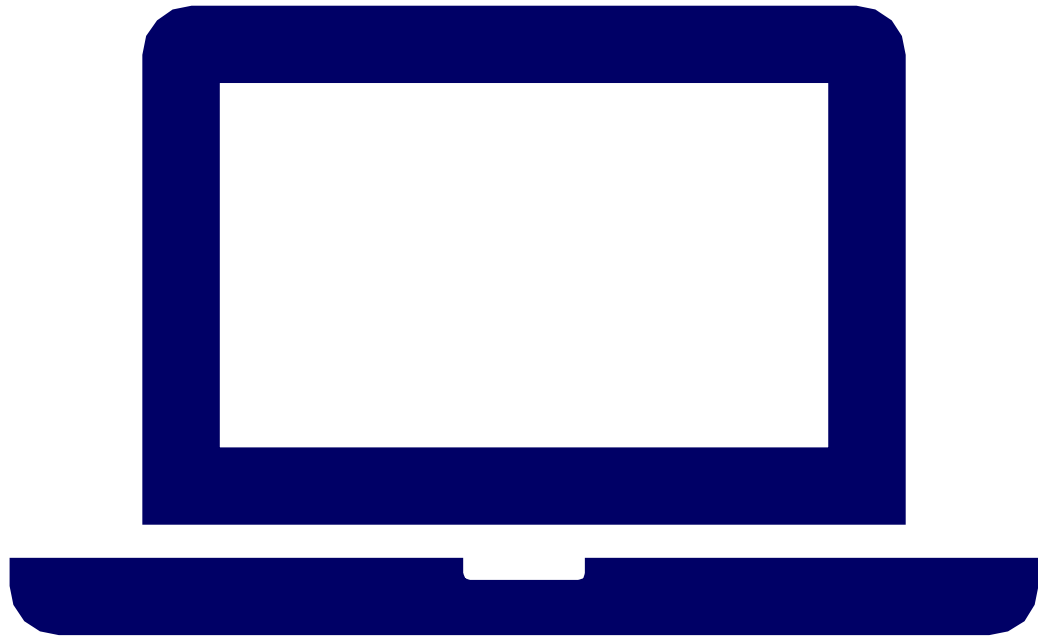
+distance(x: double, y: double): double

+distance(p: Point2D): double

+getX(): double

+getY(): double

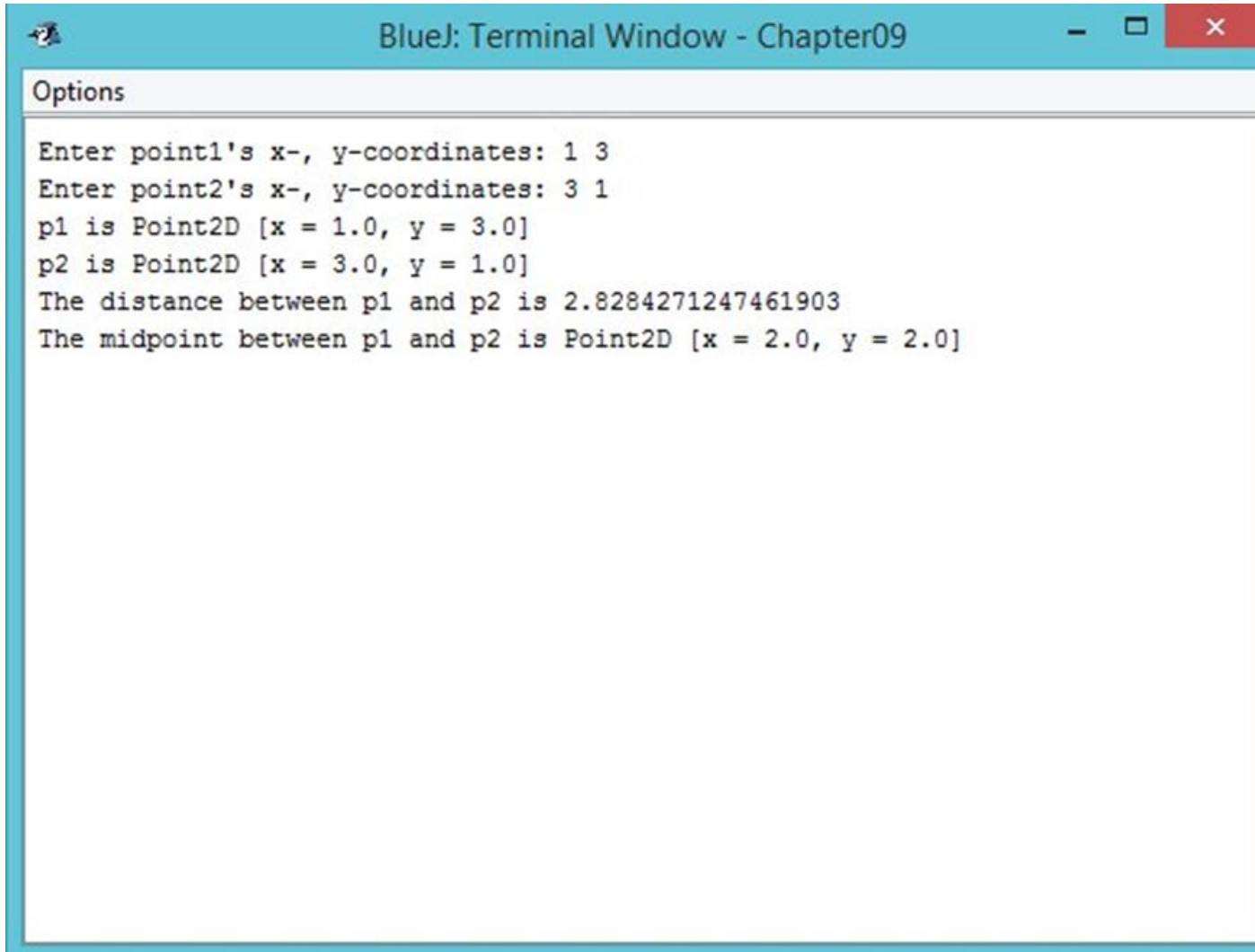
+toString(): String



# Demonstration Program

---

TESTPOINT2D.JAVA

A screenshot of a BlueJ terminal window titled "BlueJ: Terminal Window - Chapter09". The window has a light blue title bar with standard window controls (minimize, maximize, close). Below the title bar is a tab labeled "Options". The main area of the window displays the output of a Java program. The text is as follows:

```
Enter point1's x-, y-coordinates: 1 3
Enter point2's x-, y-coordinates: 3 1
p1 is Point2D [x = 1.0, y = 3.0]
p2 is Point2D [x = 3.0, y = 1.0]
The distance between p1 and p2 is 2.8284271247461903
The midpoint between p1 and p2 is Point2D [x = 2.0, y = 2.0]
```

Results:  
TestPoint2D.java



# Study the Notes

[Java\\_AWT\\_SWING\\_Javafx\\_classes.pdf](#)

---

- Learn to use packages, modules, and classes for your own programming needs. Many of the classes may not be tested in AP exam. But, knowing about them is the basis for learning programming.



# Classes in APCSA Exam

---

LECTURE 4





# AP Exam not Equal to Programming Skills

---

- AP Exam is focused on testing problem solving skills.
- Java Programming skills include problem solving skills, mastery of Java language, utilization of tools, basic computer science study and software development knowledge.



# Classes Tested in AP Computer Science

and Accessible Methods from the Java Library That May Be Included on the Exam

---

class java.lang.**Object**

class java.lang.**Integer**

class java.lang.**Double**

class java.lang.**String**

class java.lang.**Math**

class java.util.**List<E>**

class java.util.**ArrayList** implements **java.util.List** interface



# Classes Not Tested by AP Exam but Relevant to APCSA, APCSB classes

---

## **Classes:**

java.util.Scanner  
java.util.Arrays  
java.util.Random  
java.util.Collections  
java.util.Iterator  
java.lang.System  
java.lang.StringBuilder  
java.lang.Throwable  
Java.lang.Exception

## **Classes:**

java.io.File  
java.io.PrintWriter  
java.io.IOException  
java.io.EOFException

## **Packages:**

javafx package (GUI)  
java.awt package (GUI)  
java.swing package (GUI)

## **Interfaces:**

java.lang.Cloneable  
java.lang.Iterable  
java.util.Collection  
java.util.List  
java.util.Set  
java.util.Queue  
java.io.Serializable



# Information Processing

---

**Numbers, Text, Random Data(Number, Text): Covered**

**Date/Time: Not Covered**

**Graphics/Geometry: Not Covered**

**Image: GUI**

**Video: GUI**

**Audio: GUI**

## Class Hierarchy

- java.lang.**Object**
  - javafx.geometry.**Bounds**
    - javafx.geometry.**BoundingBox**
  - javafx.geometry.**Dimension2D**
  - javafx.geometry.**Insets**
  - javafx.geometry.**Point2D**
  - javafx.geometry.**Point3D**
  - javafx.geometry.**Rectangle2D**



# Data Carriers

---

## LECTURE 5



# Object is a Heterogeneous Data Record

Object is also a kind of “data carrier”

---

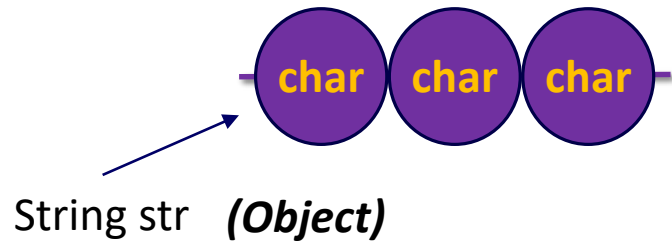
```
Class StudentGPA {  
    String name = "";  
    String ssn = "XXX-XX-XXXX";  
    String address = "";  
    int age = 15;  
    int studentID = 0;  
    int[] classCodes = new int[6];    // for 6 periods  
    ArrayList<String> classNames = new ArrayList<String>();  
    /* methods omitted*/  
}
```



# String is a collection of data but not data carrier

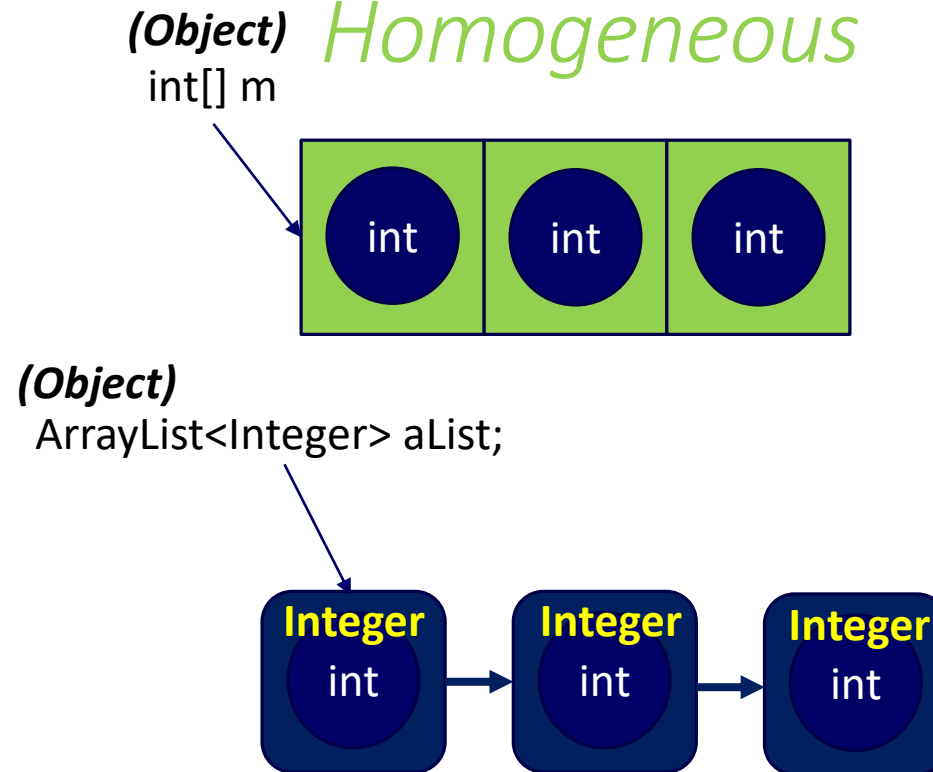
(String is immutable and can not store pointers)

## Non-Carrier

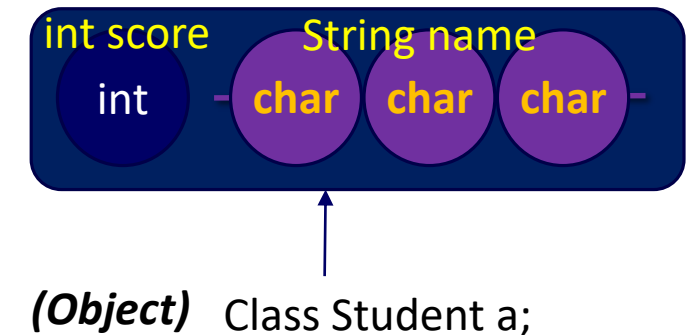


## Data Carriers

### (Object) Homogeneous



### Heterogeneous



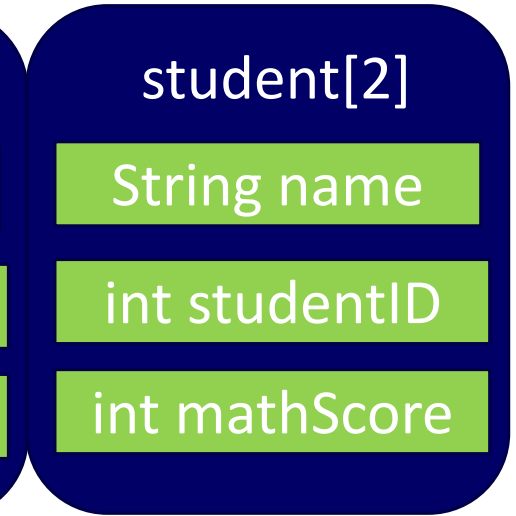
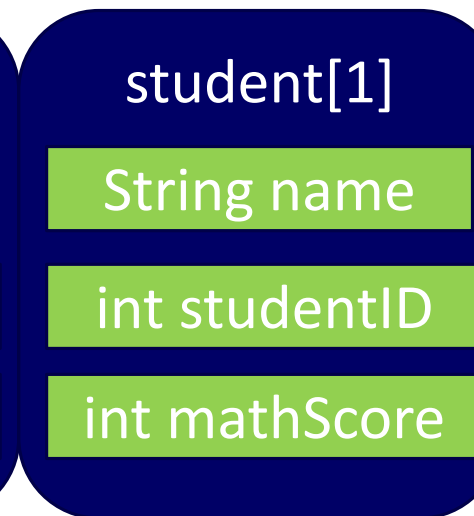
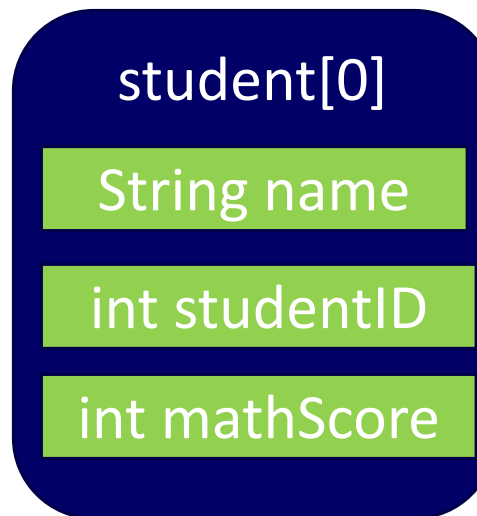
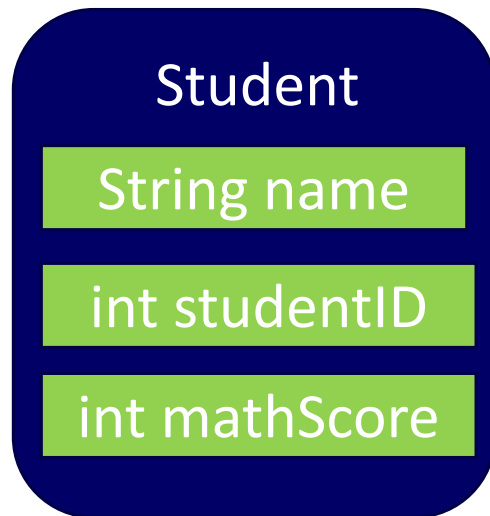




# Array of Objects

Student Class

```
Student[] students = new Student[3];
```

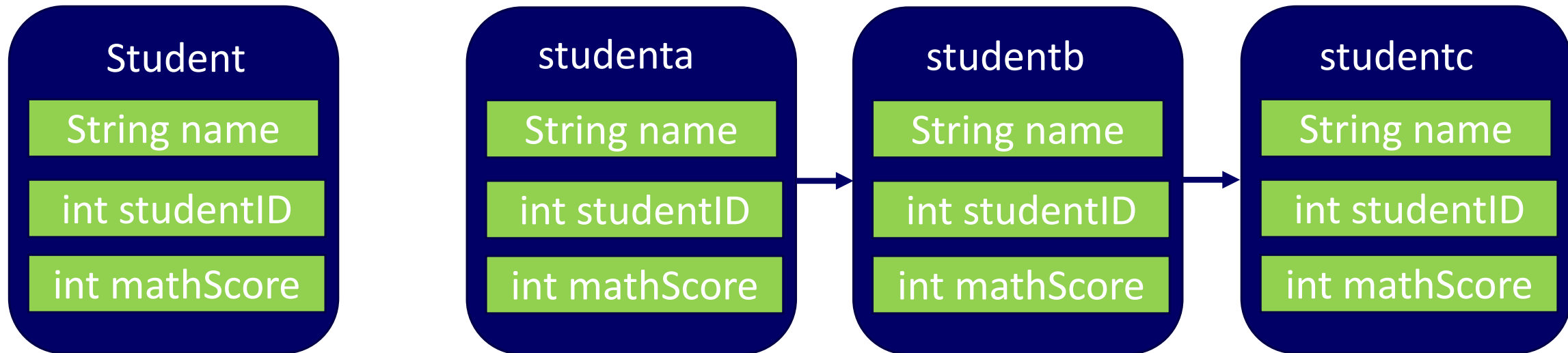




# ArrayList of Objects

Student Class

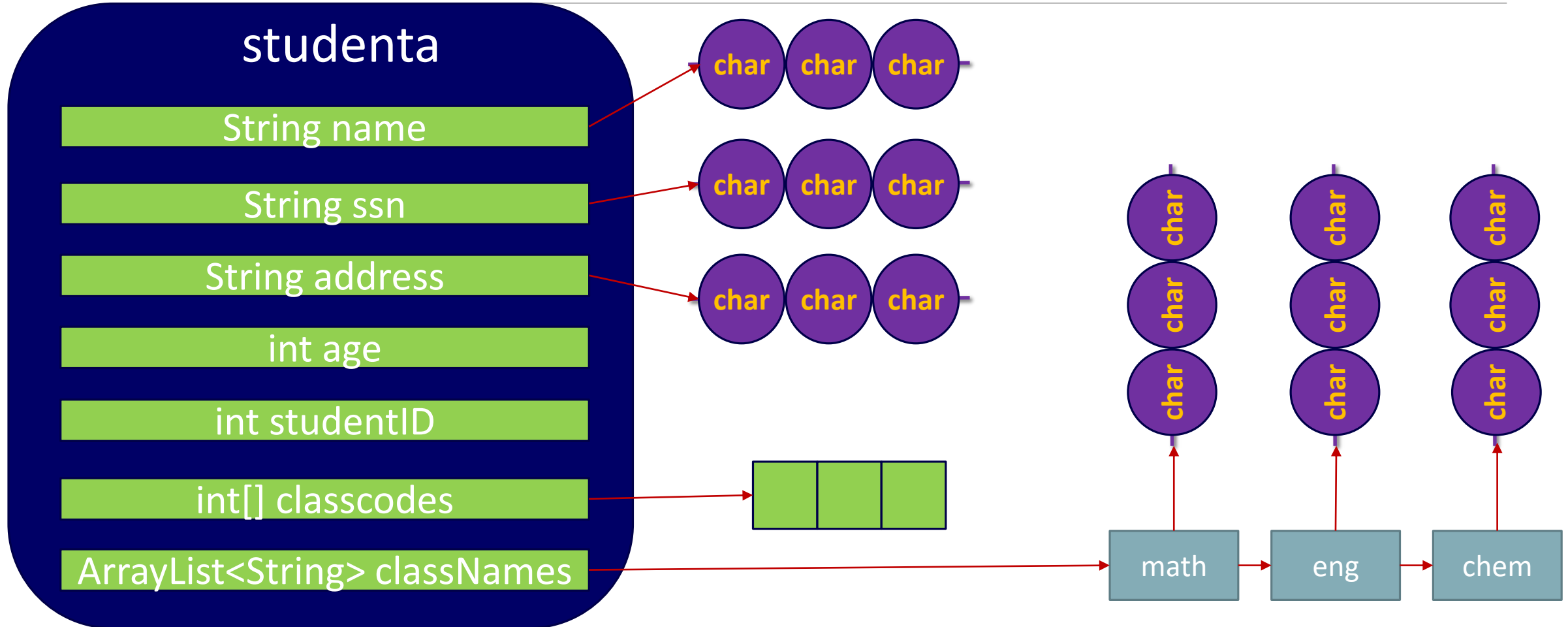
```
ArrayList<Student> al = new ArrayList<Student>();  
al.add(studenta); al.add(studentb); al.add(studentc);
```



```
Student studenta = new Student(); Student studentb = new Student(); Student studentc = new Student();
```



# Object with array and arraylist





# Demo Program: Array and ArrayList of Objects

---

LECTURE 6



# Array of Objects (ArrayList of Objects)

---

```
Circle[] circleArray = new Circle[10];  
ArrayList<Circle> circleArrayList = new ArrayList<Circle>();
```

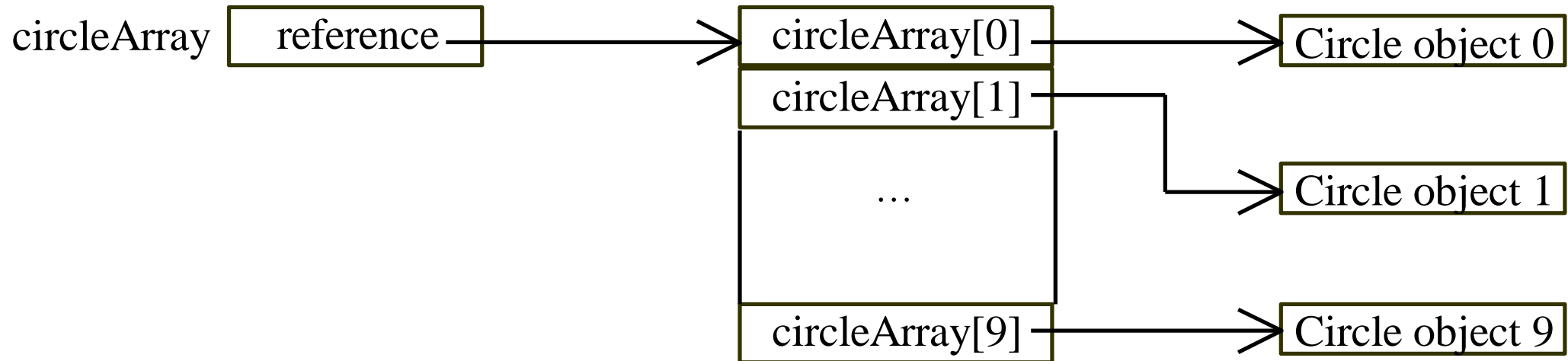
An array of objects is actually an *array of reference variables*. So invoking `circleArray[1].getArea()` involves two levels of referencing as shown in the next figure. `circleArray` references to the entire array. `circleArray[1]` references to a `Circle` object.



# Array of Objects, cont.

**Array and ArrayList are data containers/carriers**

```
Circle[] circleArray = new Circle[10];
```





# Demo Program:

Object-Oriented Version of StudentGPA series: (Washington High School)

---

(1) Integration of StudentGPA.java (Ch. 3),  
StudentInfoAnswer.java (Ch.3),  
StudentGPASimulationMode.java (Ch. 4),  
StudentGPAMethod.java (Ch. 6),  
StudentScore.java (Ch. 7),  
StudentAnswer.java (Ch. 9),  
StudentScoreMultiple.java (Ch. 9)



# Demo Program:

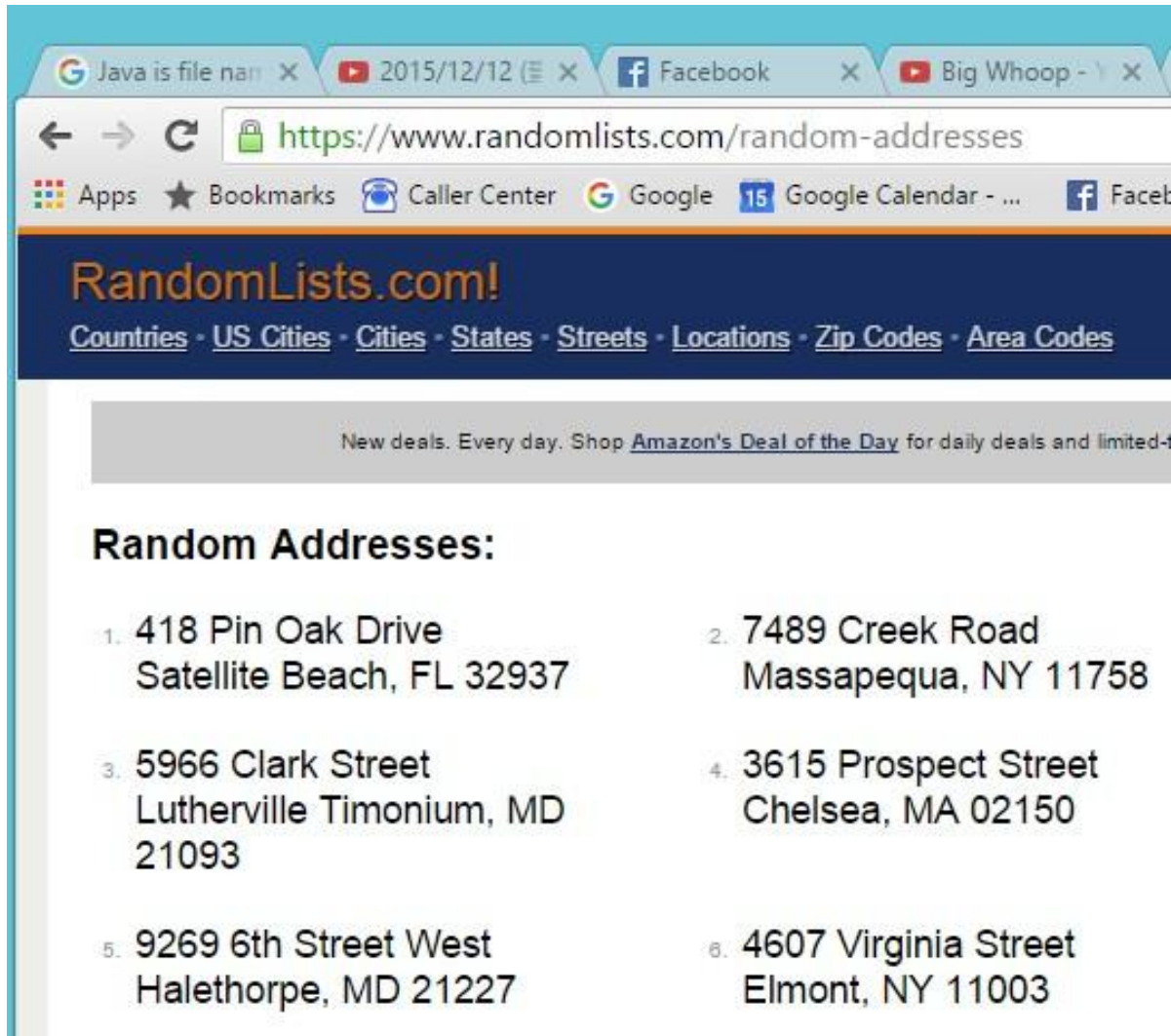
## New Features

---

(2) Newly added features:

1. Selection Manual for Student Registration Record and Class Report
2. **Data Classes** (Washington, Student, Subject, ScoreSheet)  
**Tester Classes** (Test Student, Test Subject, TestScoreSheet)  
**Random Test Pattern Generation Class**  
(RandomSheetGenerator.java Independent from Wash.)
3. Package Definition.
4. Use of Public Random Data Generators





# Public Domain Random Data Generators Random Address Generator

---

Java is file nam x 2015/12/12 x Facebook x Big

random-name-generator.info

Apps ★ Bookmarks ☎ Caller Center Google 15 Google Calendar

## Random Name Generator

Gender

Name style ☐ Common ☒ Average ☐ Rare

**Generate random names**

### Random names

1. Gilbert Medina
2. Sylvester Gross
3. Teresa Norton

# Public Domain Random Data Generators Random Name Generator

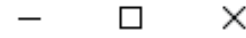
---



# Public Domain Random Data Generators Random Birthday Generator

---

BlueJ: Chapter10P2



Project Edit Tools View Help

New Class...



Compile



**DateExample**

**CalendarExample**

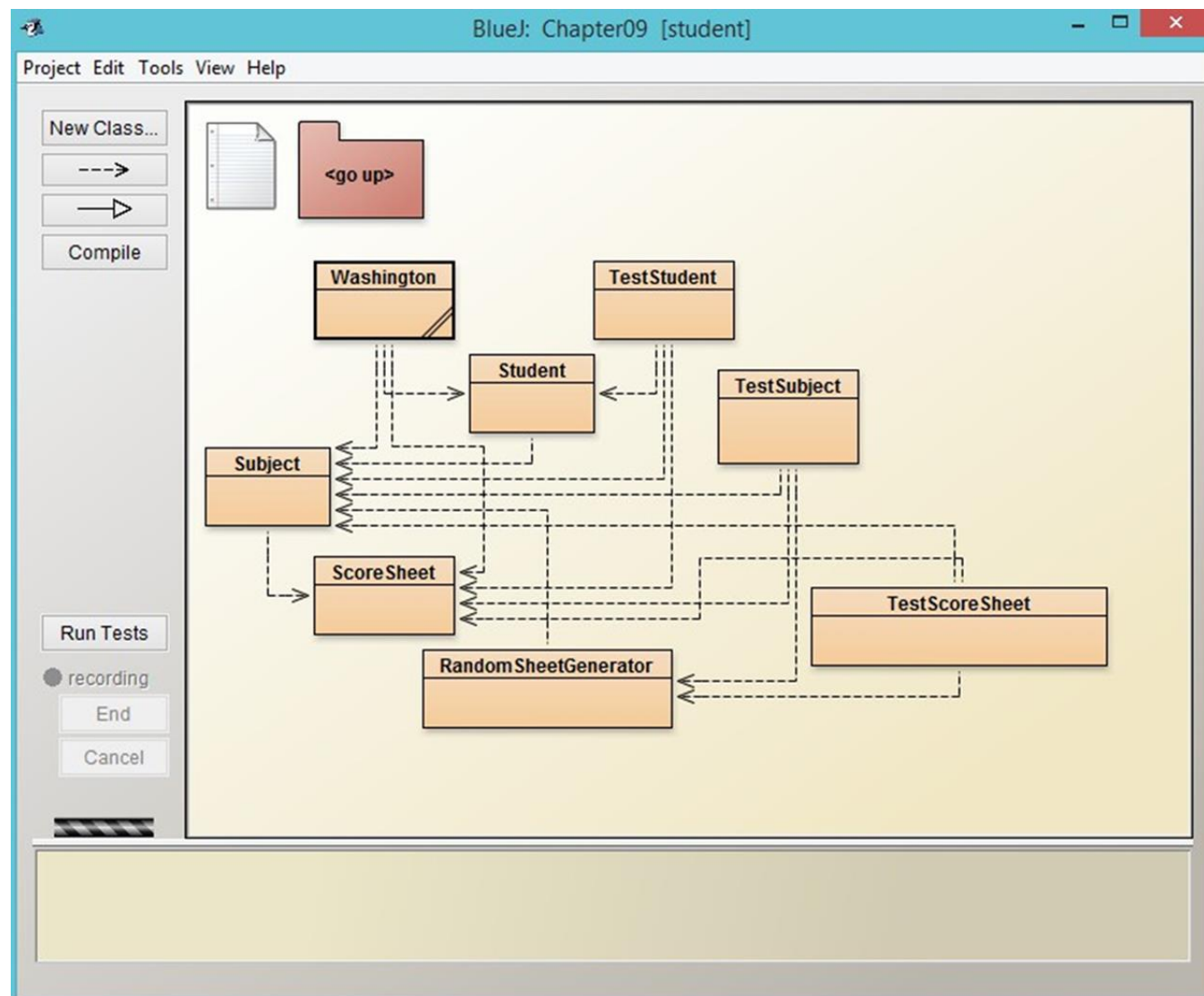
**TestPoint2D**

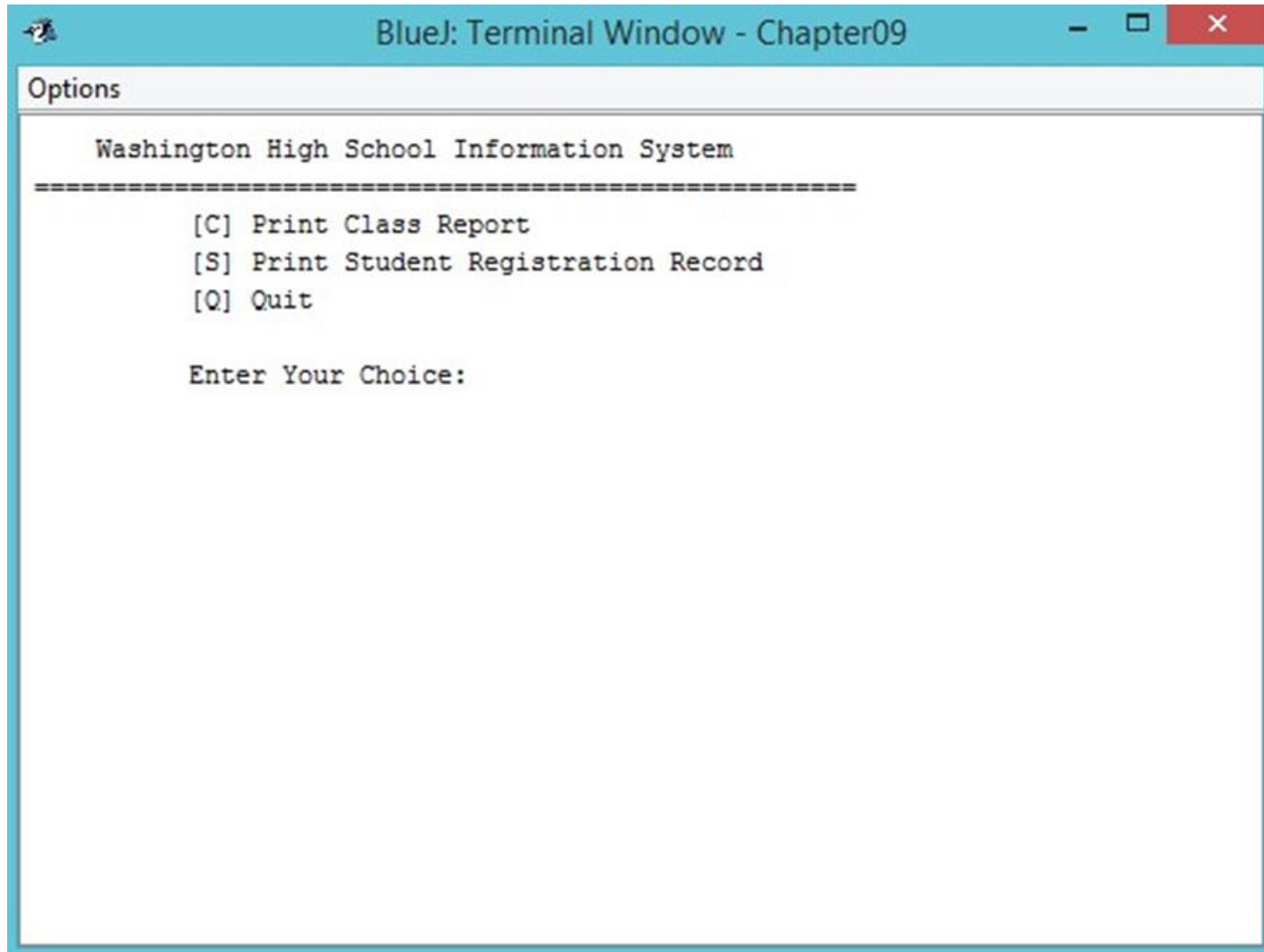
**student**



Package saved.







# Washington High School Welcome Manual

---



BlueJ: Terminal Window - Chapter09

Options

Washington High School  
Semester Class Score Report Card

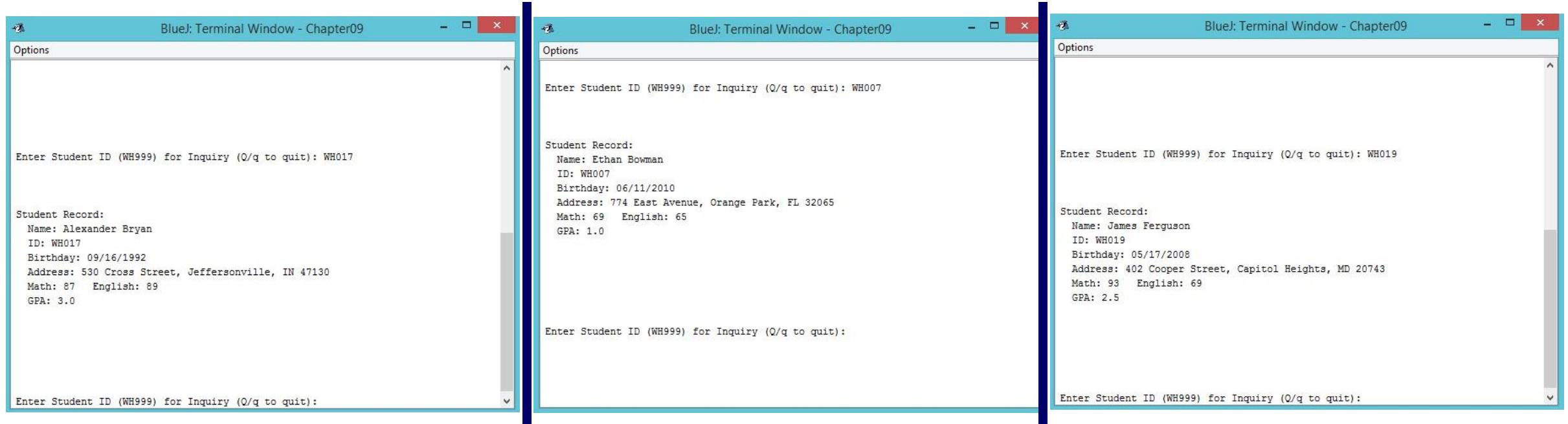
=====

ID: WH000	Name: Jackson Bryant	Math: 74 C	English: 59 F
ID: WH001	Name: Aiden Clayton	Math: 70 C	English: 67 D
ID: WH002	Name: Liam Holland	Math: 88 B	English: 80 B
ID: WH003	Name: Lucas Weber	Math: 64 D	English: 71 C
ID: WH004	Name: Noah Waters	Math: 66 D	English: 80 B
ID: WH005	Name: Mason Cannon	Math: 64 D	English: 77 C
ID: WH006	Name: Jayden Gutierrez	Math: 84 B	English: 87 B
ID: WH007	Name: Ethan Bowman	Math: 69 D	English: 65 D
ID: WH008	Name: Jacob Cummings	Math: 80 B	English: 63 D
ID: WH009	Name: Jack Kelly	Math: 77 C	English: 93 A
ID: WH010	Name: Frank Byrd	Math: 84 B	English: 75 C
ID: WH011	Name: Caden Terry	Math: 85 B	English: 76 C
ID: WH012	Name: Logan Huff	Math: 69 D	English: 71 C
ID: WH013	Name: Benjamin Riley	Math: 61 D	English: 57 F
ID: WH014	Name: Michael Henderson	Math: 79 C	English: 69 D
ID: WH015	Name: Caleb Morton	Math: 91 A	English: 66 D
ID: WH016	Name: Ryan McKinney	Math: 77 C	English: 67 D
ID: WH017	Name: Alexander Bryan	Math: 87 B	English: 89 B
ID: WH018	Name: Elijah Ford	Math: 90 A	English: 76 C
ID: WH019	Name: James Ferguson	Math: 93 A	English: 69 D
ID: WH020	Name: William Barker	Math: 72 C	English: 75 C
ID: WH021	Name: Oliver Erickson	Math: 66 D	English: 68 D
ID: WH022	Name: Connor Duncan	Math: 86 B	English: 78 C
ID: WH023	Name: Matthew Sullivan	Math: 80 B	English: 66 D
ID: WH024	Name: Daniel Allison	Math: 81 B	English: 63 D
ID: WH025	Name: Luke French	Math: 77 C	English: 86 B

Grade Distribution:

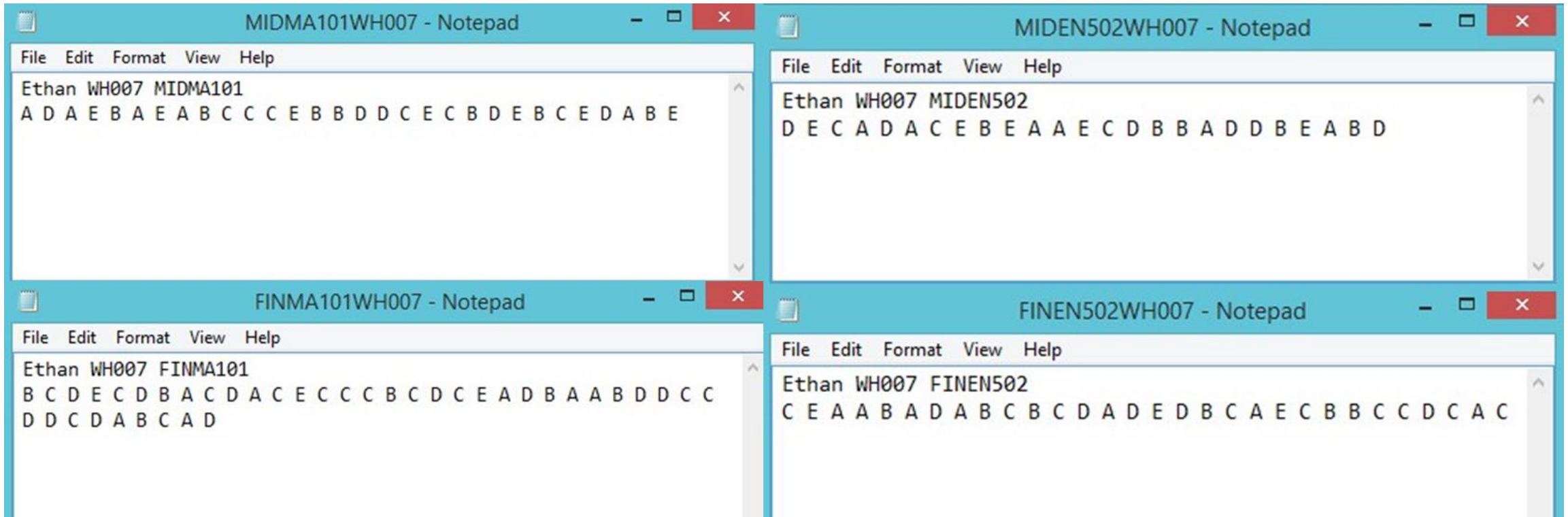
	Math Grade	English Grade
Grade A:	3	1
Grade B:	9	5
Grade C:	7	8
Grade D:	7	10
Grade F:	0	2

<<Enter any letter to Continue>>



# Student Registration Record Page





# Student Score Sheets

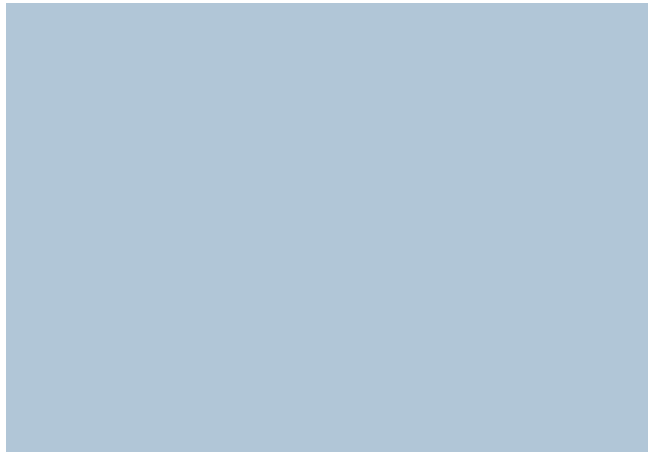
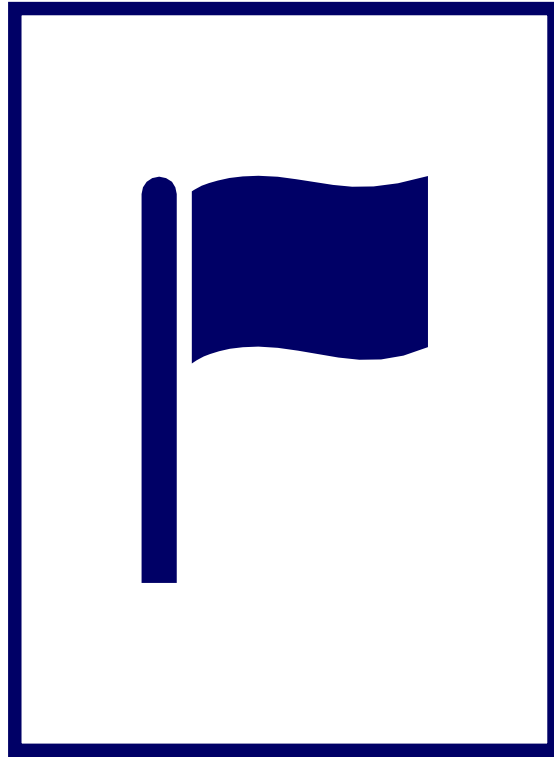
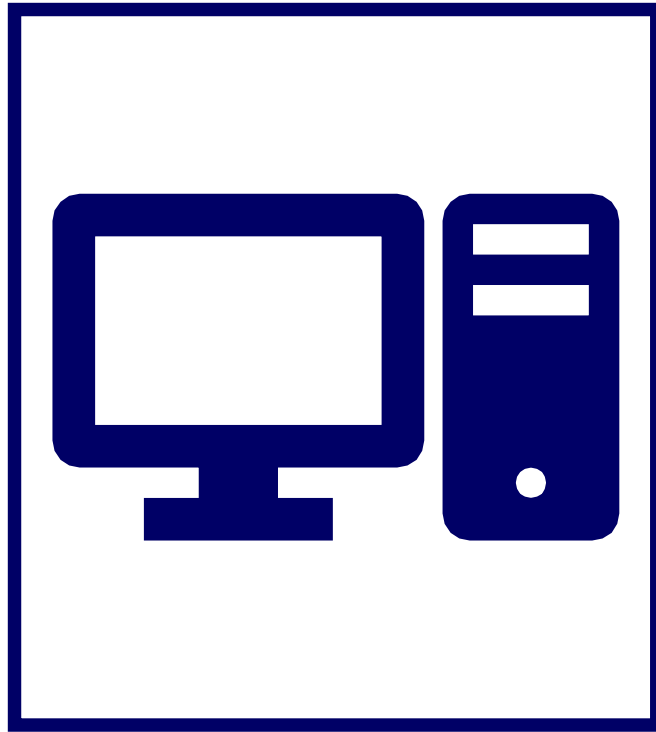
---



# Top Down Design and Bottom Up Implementation

---

- (1) Start from System Requirement of Class Score Report and Individual Student's Report Card.
- (2) Design each class' data and method calls (Decided that Student, Subject, and Score Sheets the three classes needed).
- (3) Implement from Score Sheet and Random Score Sheet Generator first. Then, Subject Class, Student Class and finally the Washington Class.

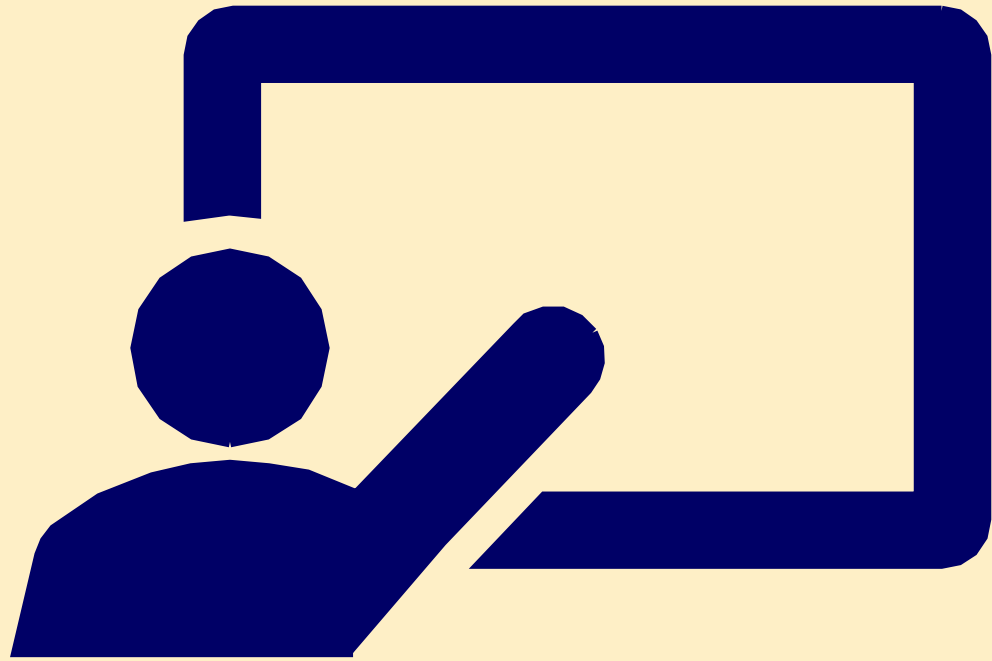


## Demo Program: Washington Project

---

Student should work on this project in Class.

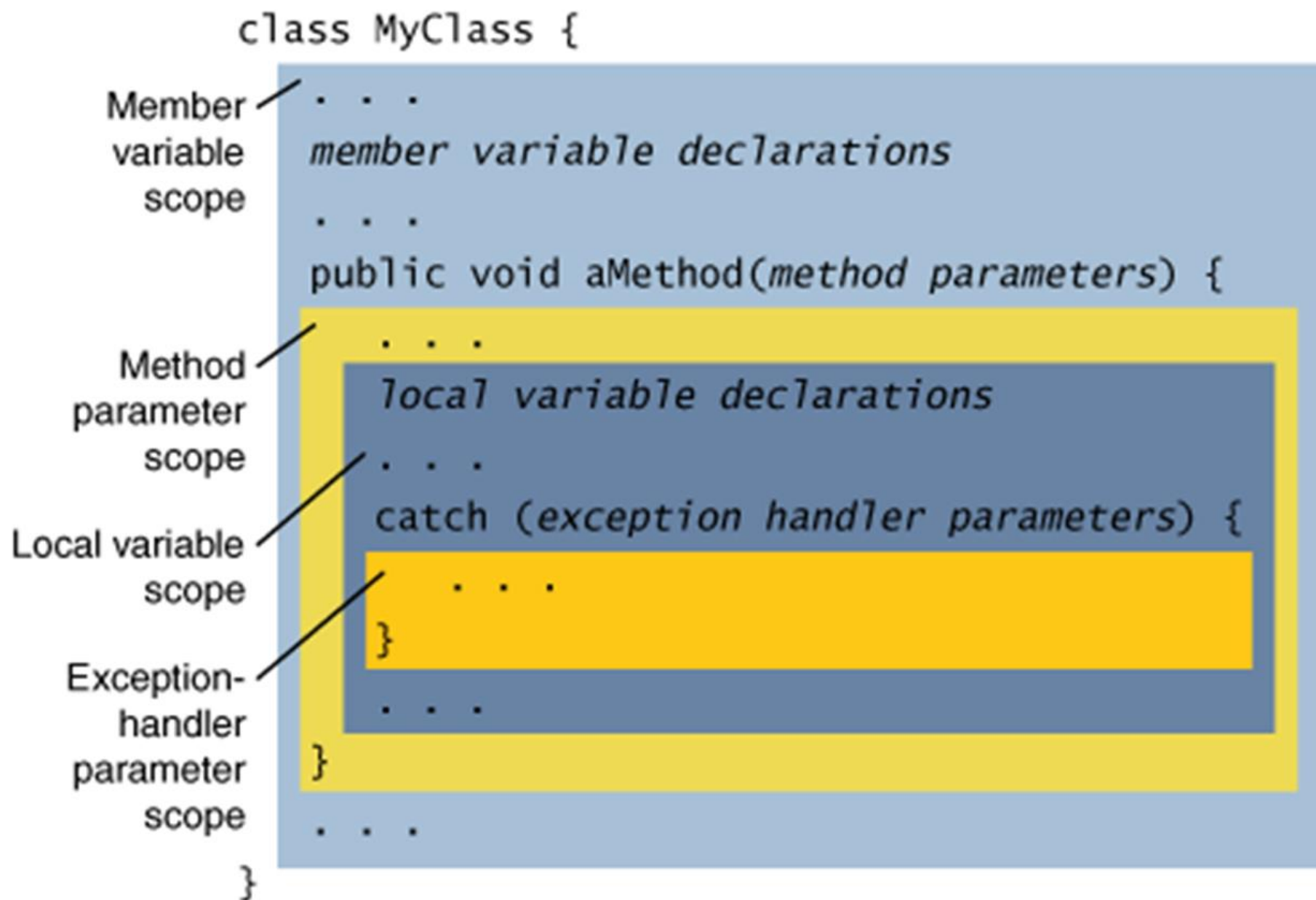
Or, as a take-out lab project.



# Scope of Members

---

LECTURE 6





# Local Variable Versus Global Variable

---

## **Global Variables:**

Member Properties:

Instance Variable - double radius;

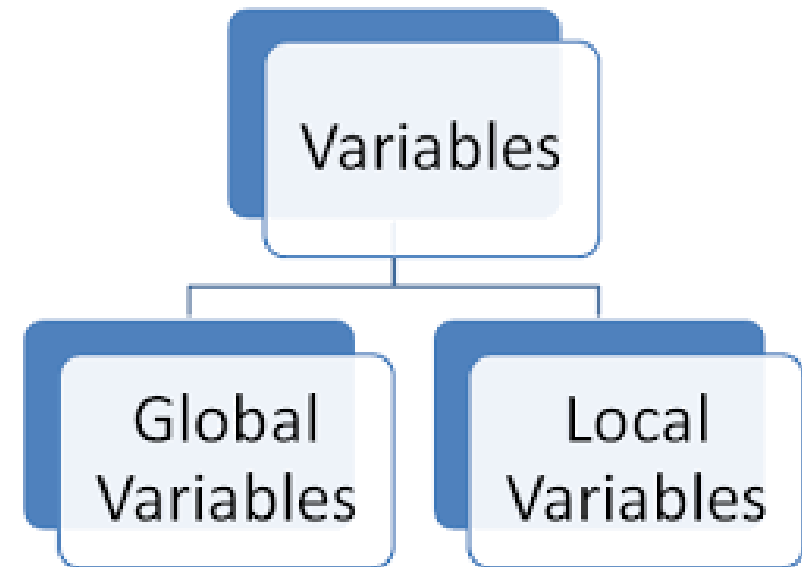
Class Variable (static) – static int num;

## **Local Variables:**

Arguments

Local Variables at Method level

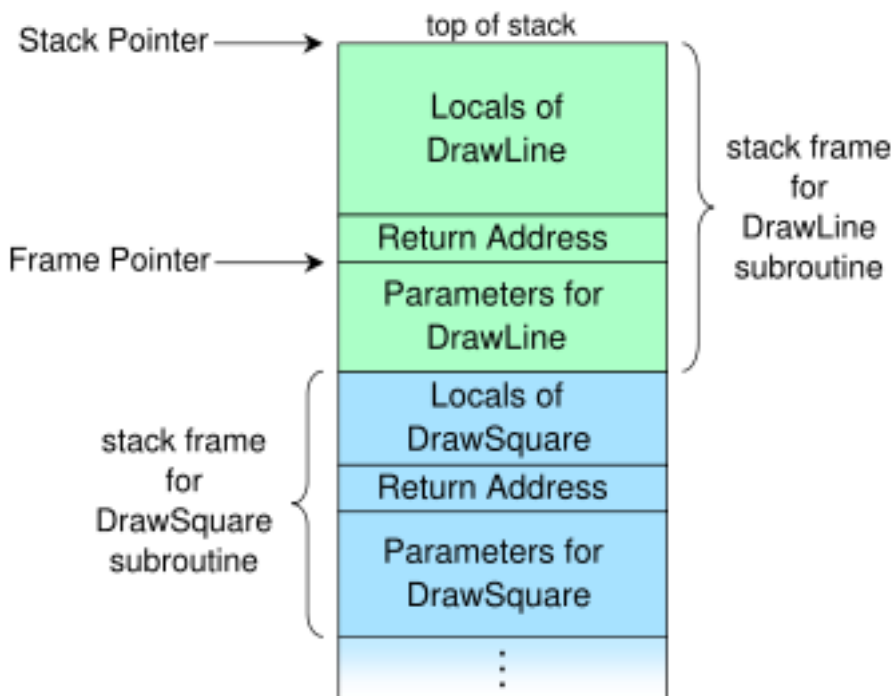
Block level local Variables



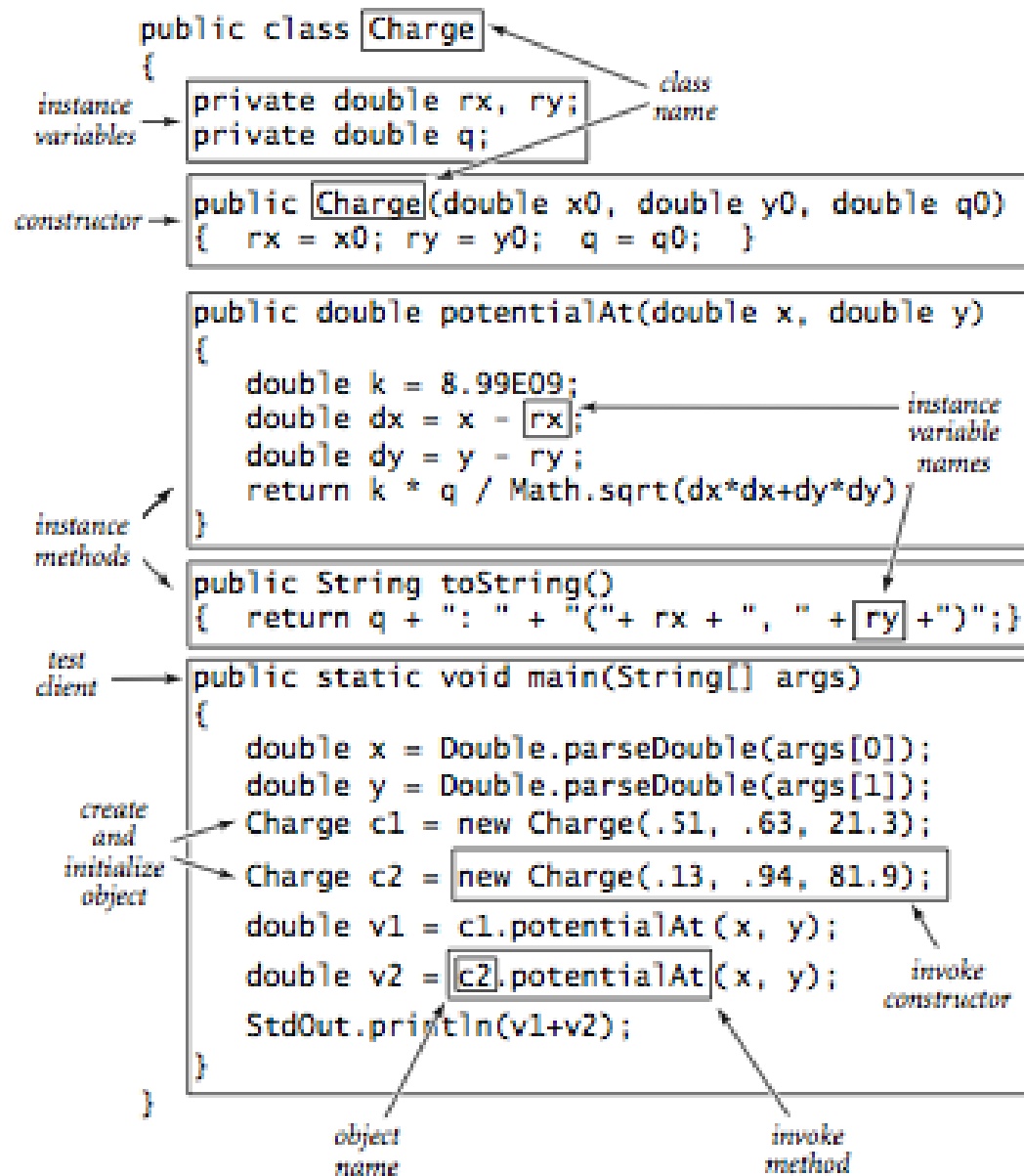


# Scope of Local Variables and Parameters

## Refer to Chapter 6: Scope of Variable (Local Variables)



- In a class definition, there are three kinds of variables:
- **instance variables** Any method in the class definition can access these variables **(is global, not local)**
  - **parameter variables** Only the method where the parameter appears can access these variables. This is how information is passed to the object.
  - **local variables** Only the method where the parameter appears can access these variables. These variables are used to store intermediate results.



Anatomy of a class

# Instance Members

- Instance Variables
- Instance Methods

Static Members (Class Members) can also be used as Instance Members. (call from instance is valid) Class Method can not call instance methods.





```
public class Charge()
{
    private double rx, ry;
    private double q;
    .
    .
}
```

*Instance variables*

*instance variable declarations*

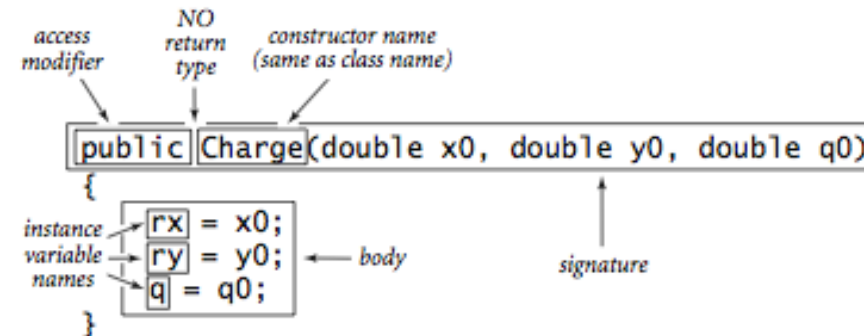
# Instance Variables

- To write code for the methods that manipulate data type values, the first thing that we need is to declare variables that we can use to refer to the values in code. These variables can be any type of data. We declare the types and names of these instance variables in the same way as we declare local variables.
- There is a critical distinction between **instance variables** and the local variables within a **static** method or a block that you are accustomed to using: there is just one value corresponding to each local variable name, but there are numerous values corresponding to each instance variable (one for each object that is an instance of the data type). Therefore, **static methods** cannot access **instance variables**. (instance variables may not be available)

# Constructors

## Create Instances

- A constructor creates an object and provides a reference to that object. Java automatically invokes a constructor when a client program uses the keyword `new`. Java does most of the work: our code only needs to initialize the instance variables to meaningful values. Constructors always share the same name as the class. To the client, the combination of `new` followed by a constructor name (with argument values enclosed within parentheses) is the same as a function call that returns a value of the corresponding type.

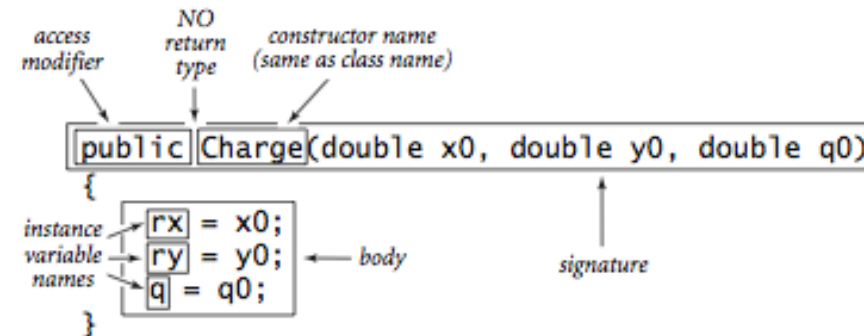


*Anatomy of a constructor*

# Constructors

## Create Instances

- A **constructor signature** has **no** return type because constructors always return a reference to an object of its data type. Each time that a client invokes a constructor) Java automatically
  - allocates memory space for the object
  - invokes the constructor code to initialize the data type values
  - returns a reference to the object

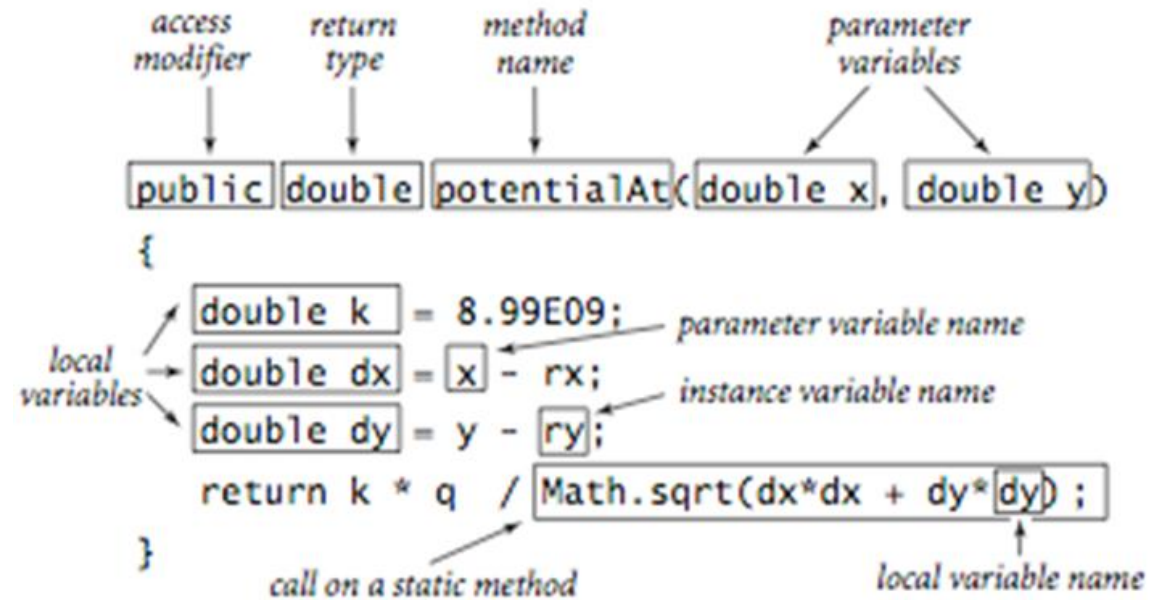


*Anatomy of a constructor*

# Instance methods

in a class with static modifier

- Each instance method has a signature (which specifies its return type and the types and names of its parameter variables) and a body (which consists of a sequence of statements, including a return statement that provides a value of the return type back to the client).

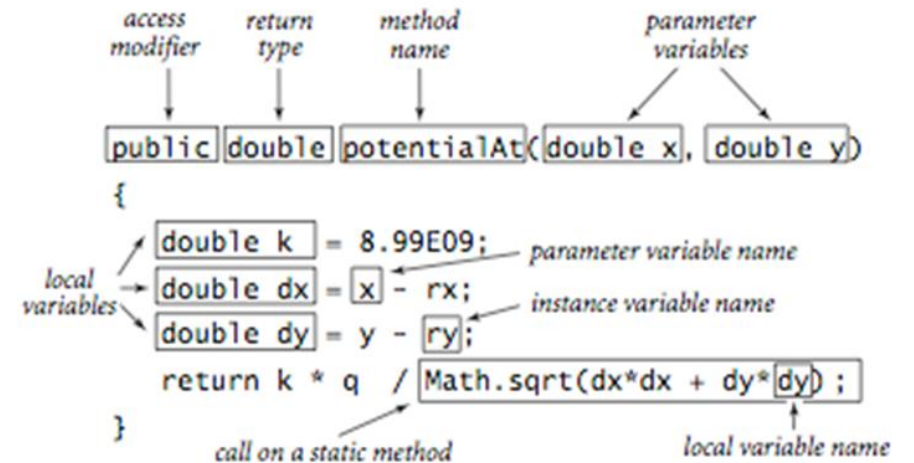


Anatomy of a data-type method

# Instance methods

in a class with static modifier

- When a client invokes a method, the parameter values are initialized with client values, the lines of code are executed until a return value is computed, and the value is returned to the client, with the same effect as if the method invocation in the client were replaced with that value. All of this action is the same as for static methods, but there is one critical distinction for instance methods: **they can perform operations on instance values.**



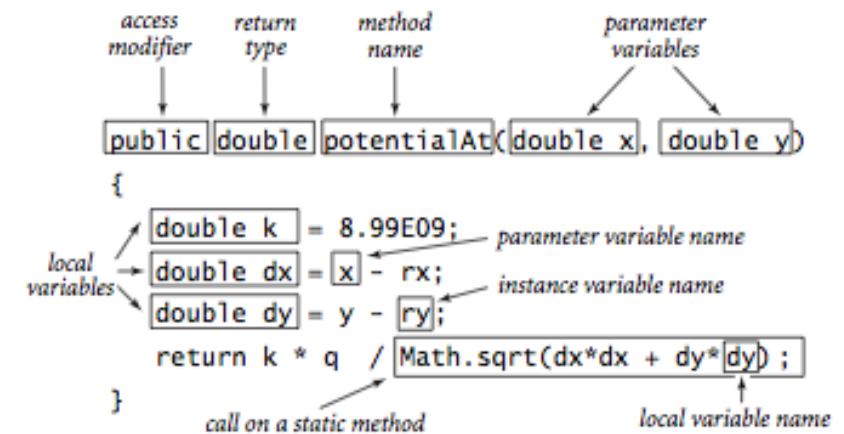
Anatomy of a data-type method



# Instance methods

## (methods in a class with static modifier)

- When a client invokes a method, the parameter values are initialized with client values, the lines of code are executed until a return value is computed, and the value is returned to the client, with the same effect as if the method invocation in the client were replaced with that value. All of this action is the same as for static methods, but there is one critical distinction for instance methods: **they can perform operations on instance values.**



Anatomy of a data-type method



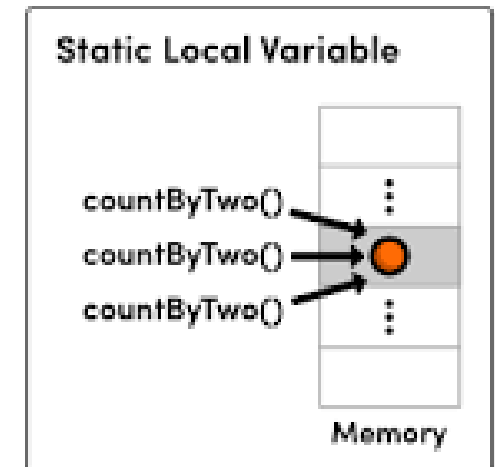
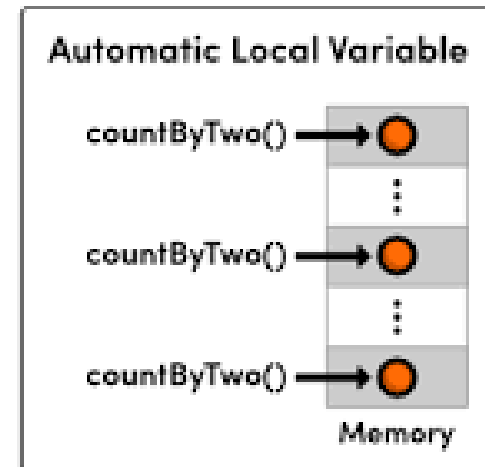
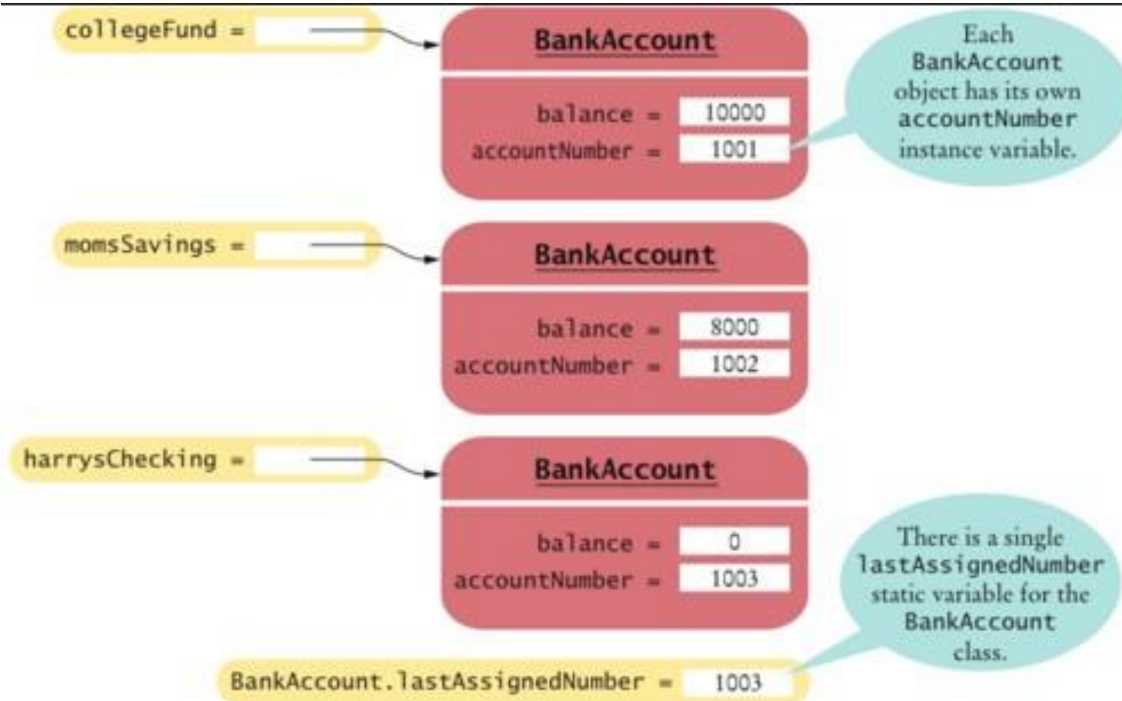
# Summary of Local Variables

---

<i>variable</i>	<i>purpose</i>	<i>example</i>	<i>scope</i>
instance	data-type value	rx	class
parameter	pass value from client to method	x	method
local	temporary use within method	dx	block



# Static Members



*Static Methods (Refers to Chapter 6)*

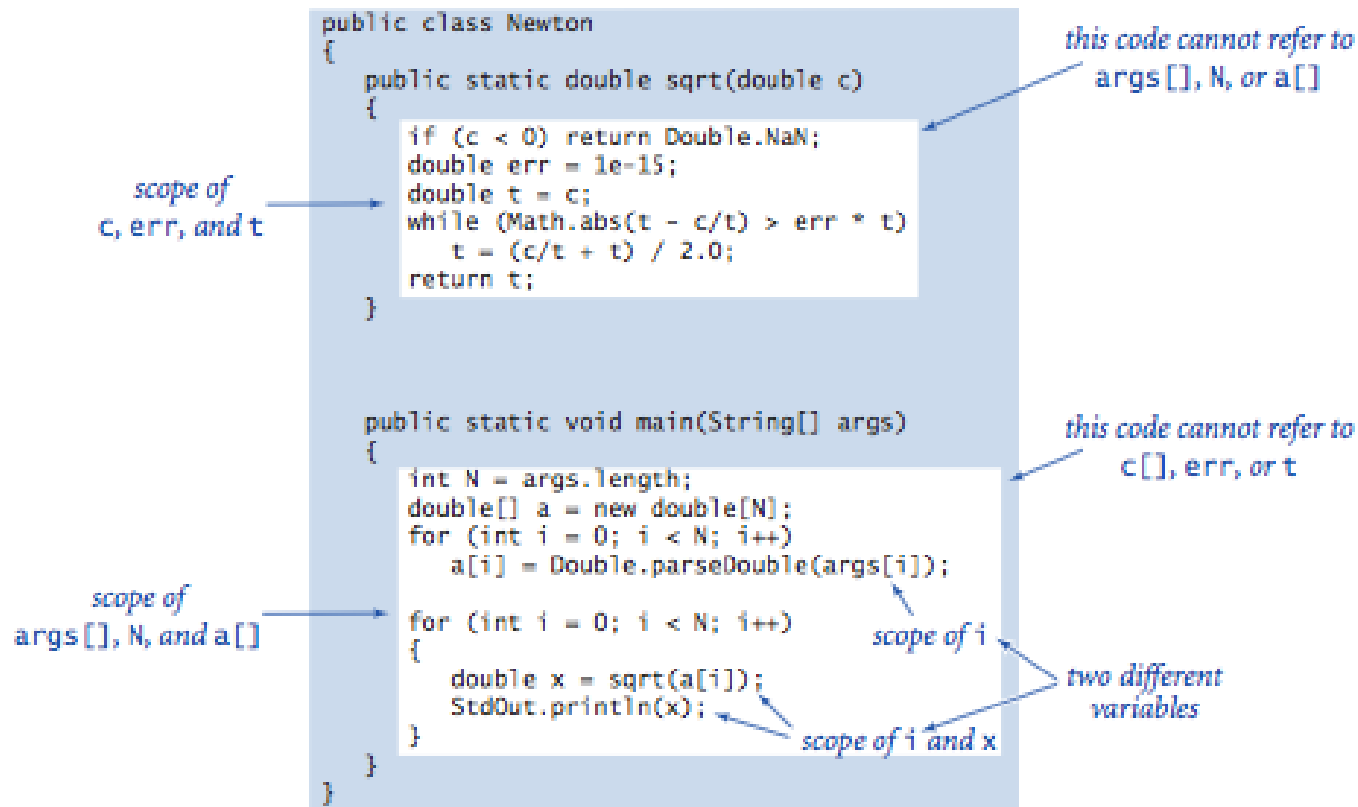
*Static Variables (Refers to the Classes and Objects (1) in this Chapter 9)*





# Static Methods

(Program control structure Only, not related to data)



Scope of local and argument variables

- The use of static methods is easy to understand. For example, when you write `Math.abs(a-b)` in a program, the effect is as if you were to replace that code by the value that is computed by Java's `Math.abs()` method when presented with the value `a-b`.
- If you think about what the computer has to do to create this effect, you will realize that it involves changing a program's **flow of control**.



# Lab:

class (static) Variables and  
Methods in Complex class

---

LECTURE 7



# Lab Project

---

1. Create a project named a3. Copy the Complex.java Class from a2 and, then, modify the program from there.
2. Keep the instance method and create another method implementation in static method format except the constructors, the getter (accessor) methods and (setter) mutator methods.
3. Try to open two files for output. Output the output of the instance methods to “ComplexTestInstance.txt” and the output of the static methods to “ComplexTestStatic.txt”.



# Example (Conversion):

```
public Complex add(Complex cc){  
    Complex result = new Complex();  
    result.r = this.r + cc.r;  
    result.i = this.i + cc.i;  
    return result;  
}
```

reference data type to current object

```
public static Complex add(Complex c1, Complex c2){  
    Complex result = new Complex();  
    result.r = c1.r + c2.r;  
    result.i = c1.i + c2.i;  
    return result;  
}
```

parameter Variable



# In Tester Class:

## ClassName.staticMethod(op1, op2)

---

```
System.out.println("Addition c4=c1-c2: ");
Complex c4 = Complex.minus(c1, c2);
System.out.println("c4 Real:    " + c4.getR());
System.out.println("c4 Imaginary: " + c4.getI());
System.out.println();
System.out.println("Negation of c4: ");
System.out.println("-c4:    " + Complex.toString(Complex.neg(c4)));
System.out.println();
System.out.println("Conjugate of c4: ");
System.out.println("Conj(c4): " + Complex.toString(Complex.conjugate(c4)));
System.out.println();
System.out.println("Inverse of c4: ");
System.out.println("Inv(c4): " + Complex.toString2(Complex.inverse(c4)));
System.out.println();
```



# Lab

---

A3.ZIP

Finish your own version first. Download the a3.zip. Unzip it and copy it to a certain directory. The directory also contains some execution results.



# Visibility Modifiers

---

LECTURE 8



# Visibility Modifiers and Accessor/Mutator Methods

---

- By default, the class, variable, or method can be accessed by any class in the same package.

## **public**

The class, data, or method is visible to any class in any package.

## **protected**

The class, data, or method is visible to any sub-class of this class in any package.

## **private**

The data or methods can be accessed only by the declaring class.

- The get and set methods are used to read and modify private properties.





# Data and Methods Visibility

Modifier on Members in a class	Accessed from the same class	Accessed from the same package	Accessed from a subclass in a different package	Accessed from a different package
public	O	O	O	O
protected	O	O	O	X
default	O	O	X	X
private	O	X	X	X

package p1;

```
public class C1 {  
    public int x;  
    int y;  
    private int z;  
  
    public void m1() {  
    }  
    void m2() {  
    }  
    private void m3() {  
    }  
}
```

```
public class C2 {  
    void aMethod() {  
        C1 o = new C1();  
        can access o.x;  
        can access o.y;  
        cannot access o.z;  
  
        can invoke o.m1();  
        can invoke o.m2();  
        cannot invoke o.m3();  
    }  
}
```

package p2;

```
public class C3 {  
    void aMethod() {  
        C1 o = new C1();  
        can access o.x;  
        cannot access o.y;  
        cannot access o.z;  
  
        can invoke o.m1();  
        cannot invoke o.m2();  
        cannot invoke o.m3();  
    }  
}
```

package p1;

```
class C1 {  
    ...  
}
```

```
public class C2 {  
    can access C1  
}
```

package p2;

```
public class C3 {  
    cannot access C1;  
    can access C2;  
}
```

The private modifier restricts access to within a class, the default modifier restricts access to within a package, and the public modifier enables unrestricted access.



# NOTE

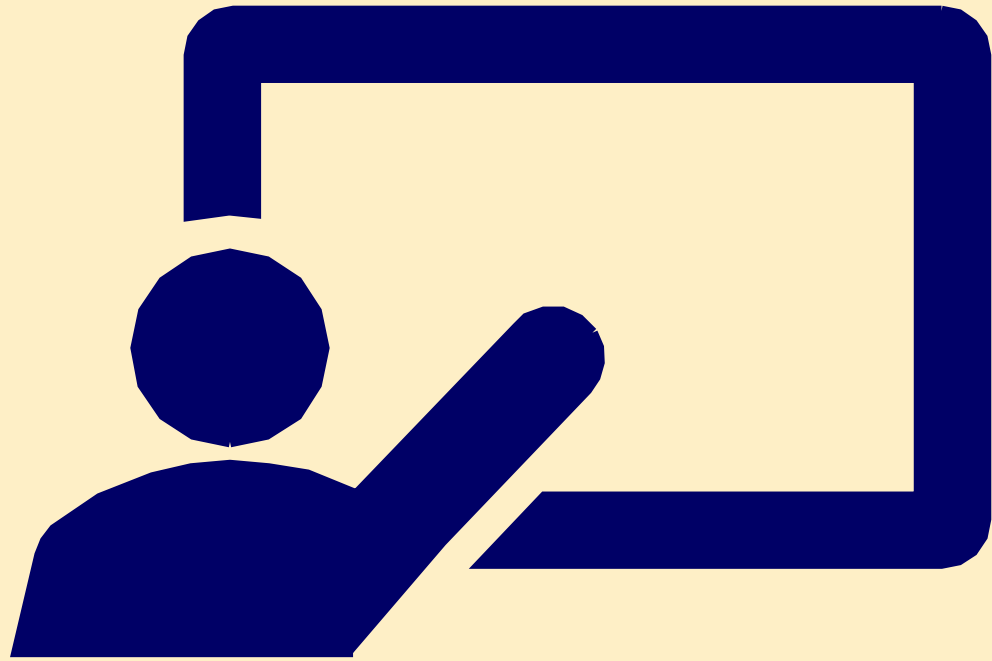
An object cannot access its private members, as shown in (b). It is OK, however, if the object is declared in its own class, as shown in (a).

```
public class F {  
    private boolean x;  
  
    public static void main(String[] args) {  
        F f = new F ();  
        System.out.println(f.x);  
        System.out.println(f.convert());  
    }  
  
    private int convert(boolean b) {  
        return x ? 1 : -1;  
    }  
}
```

(a) This is OK because object f is used inside the F class

```
public class Test {  
    public static void main(String[] args) {  
        Foo f = new F();  
        System.out.println(f.x);  
        System.out.println(f.convert(f.x));  
    }  
}
```

(b) This is wrong because x and convert are private in F.



# Complex.Complex Generation of Complex.jar file and Installation

---

LECTURE 9



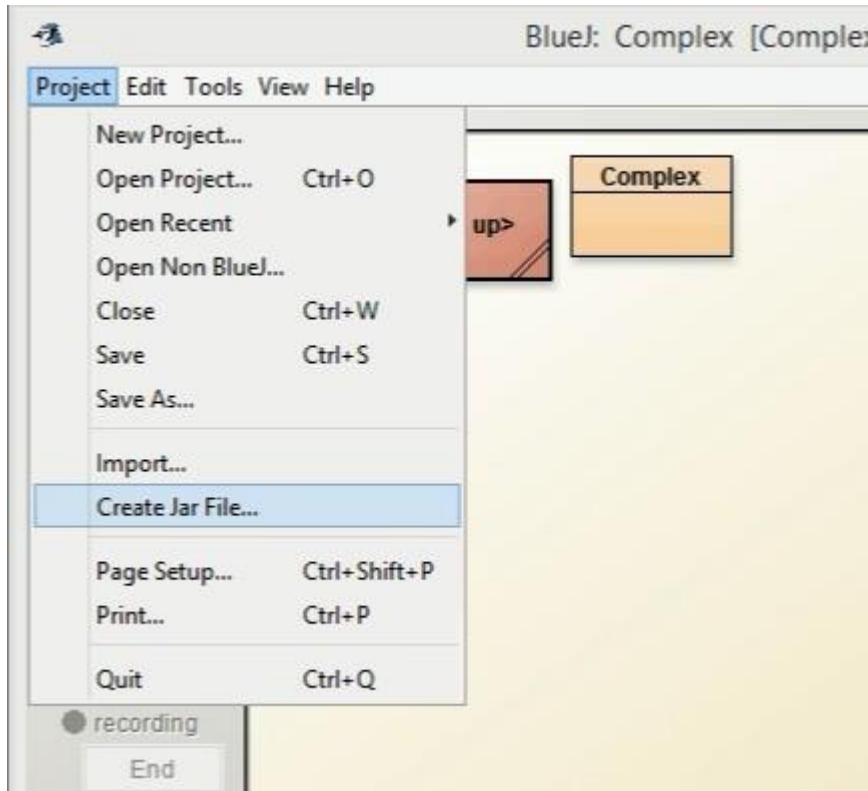
# Complex.Complex

---

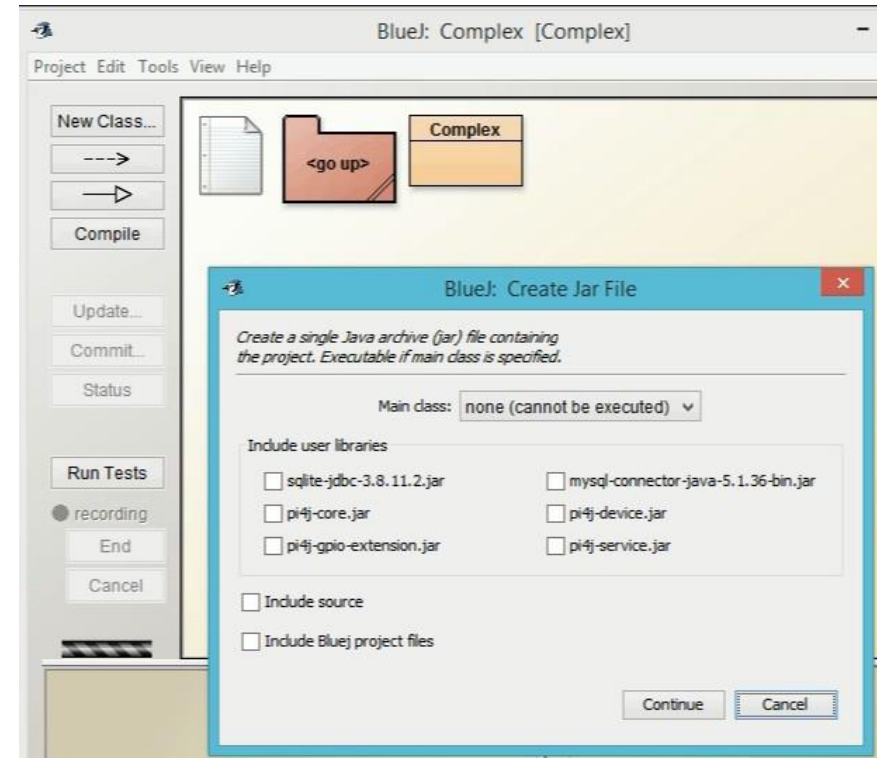
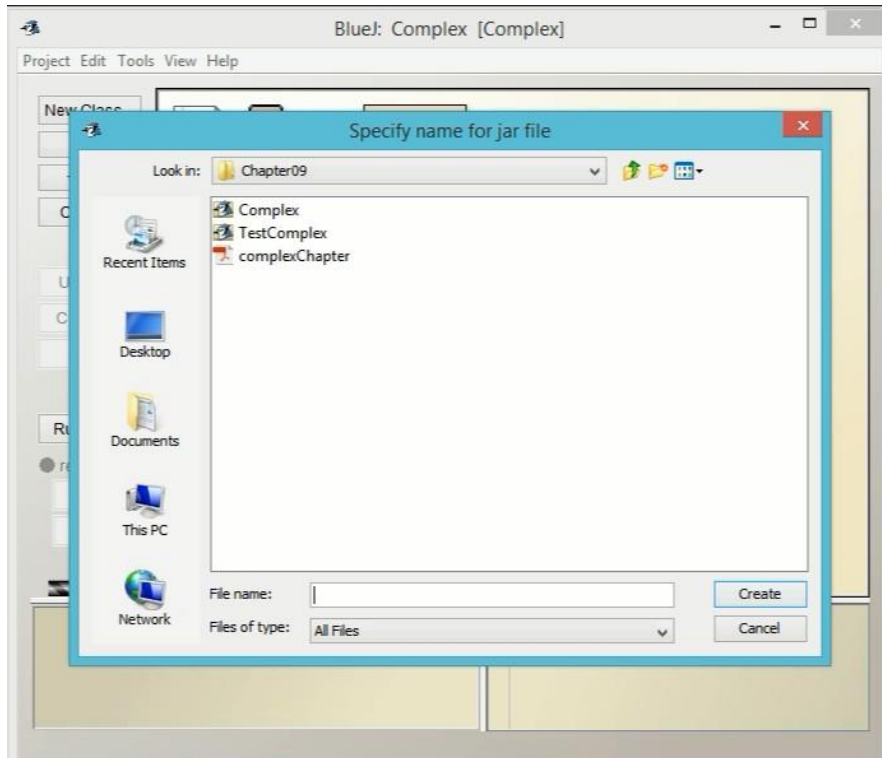
- In order to create a mathematical library which provides all of the data type and methods for complex numbers. A **Complex** package is created.
- First **Complex** is a package. The second Complex is a class (module).
- In the future, we can also create **Complex.Phasor** (for Polar Coordinates phasor), **Complex.Vector** (for Cartesian Coordinates Vector), and other utility classes for **Complex** number and related mathematical program methods.
- We start to explain how to put the complex library package (.jar file) in to BlueJ system.



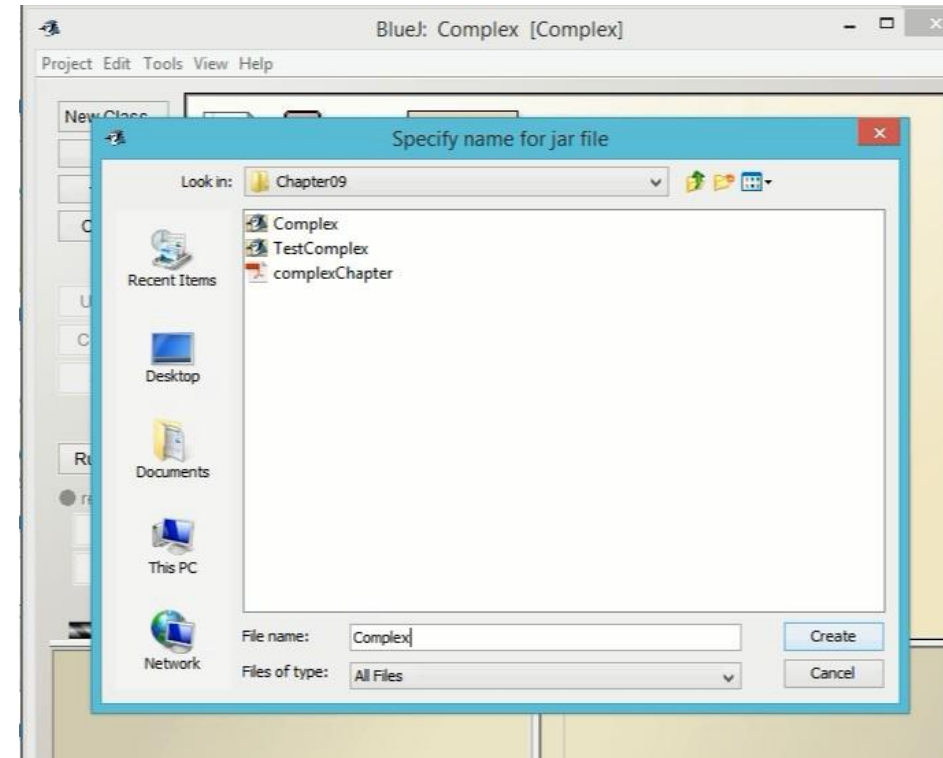
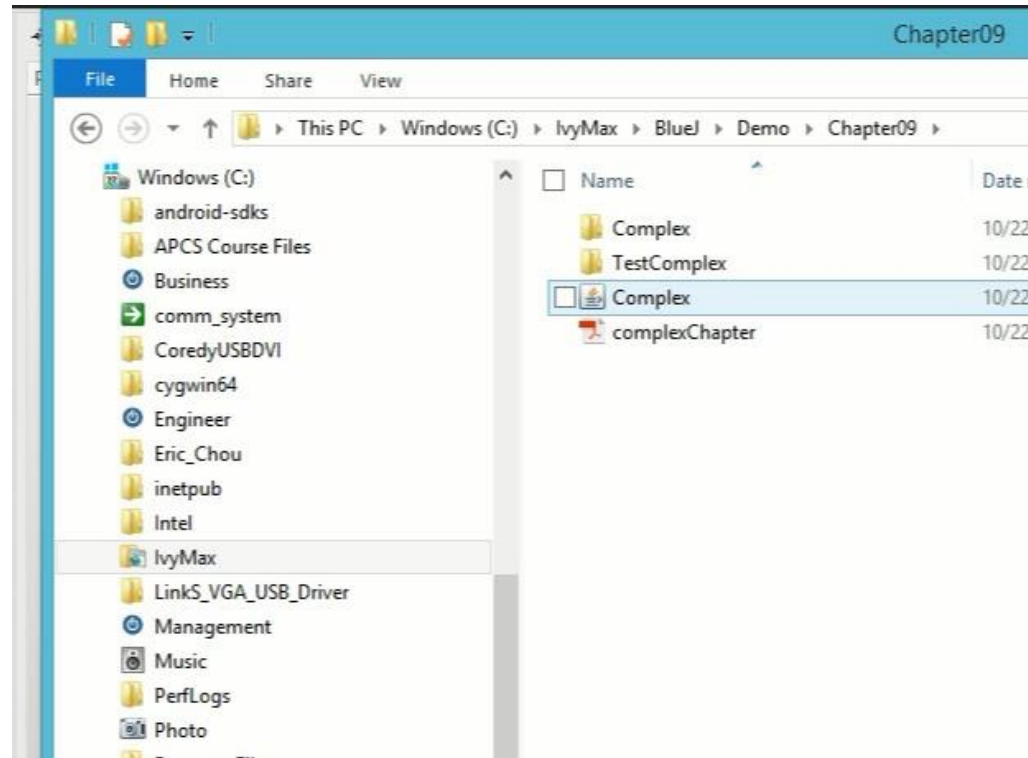
After the Complex package and Complex Class is created. How is now to create a .jar file.



At package view, click **Project**, click **Create Jar File**.

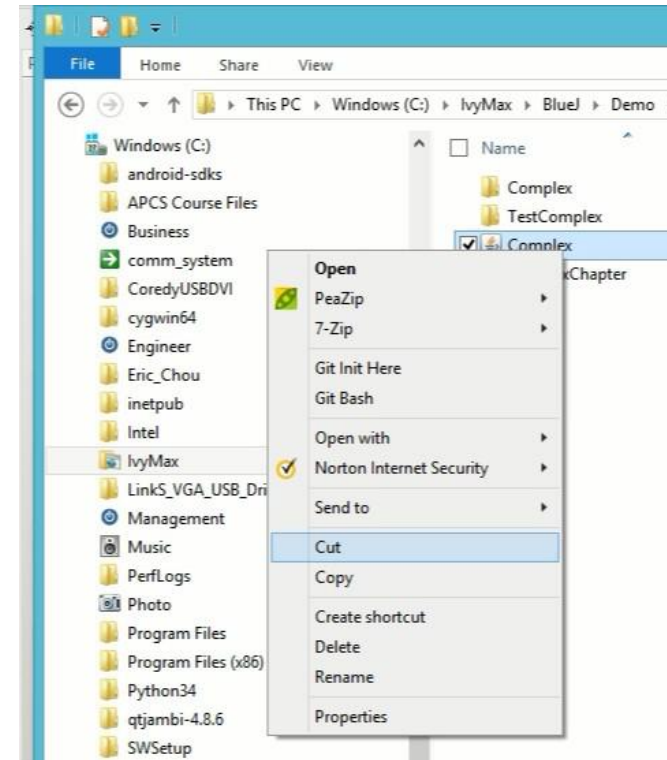
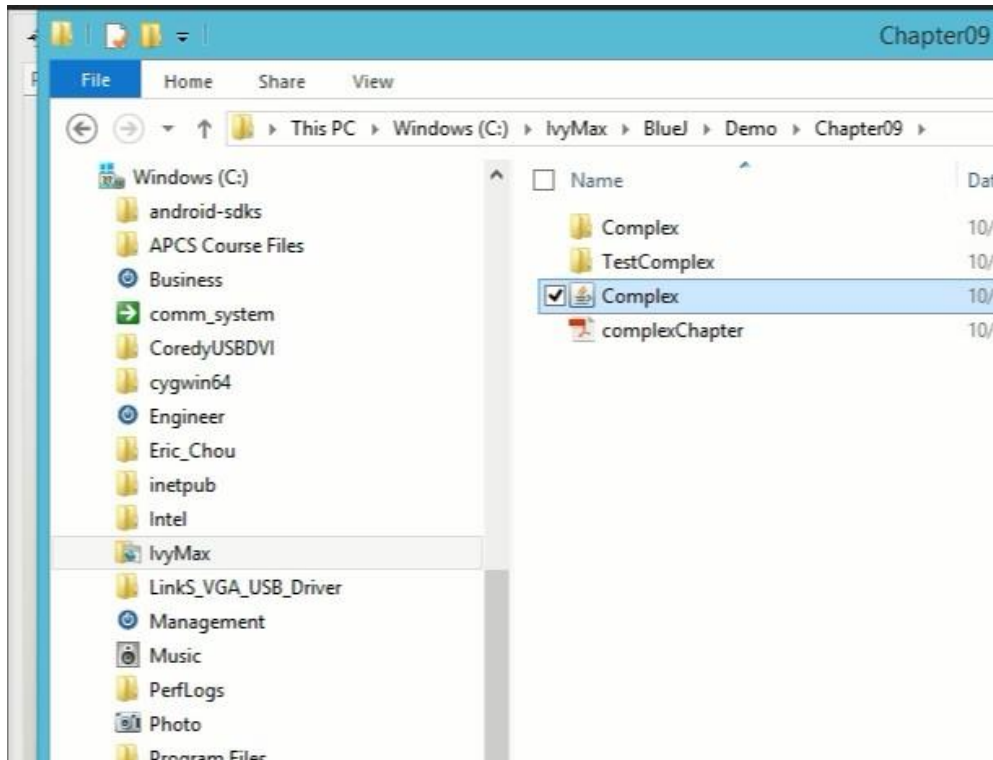


Pick proper Jar File Setting (currently just continue.) Then, specify the name as Complex

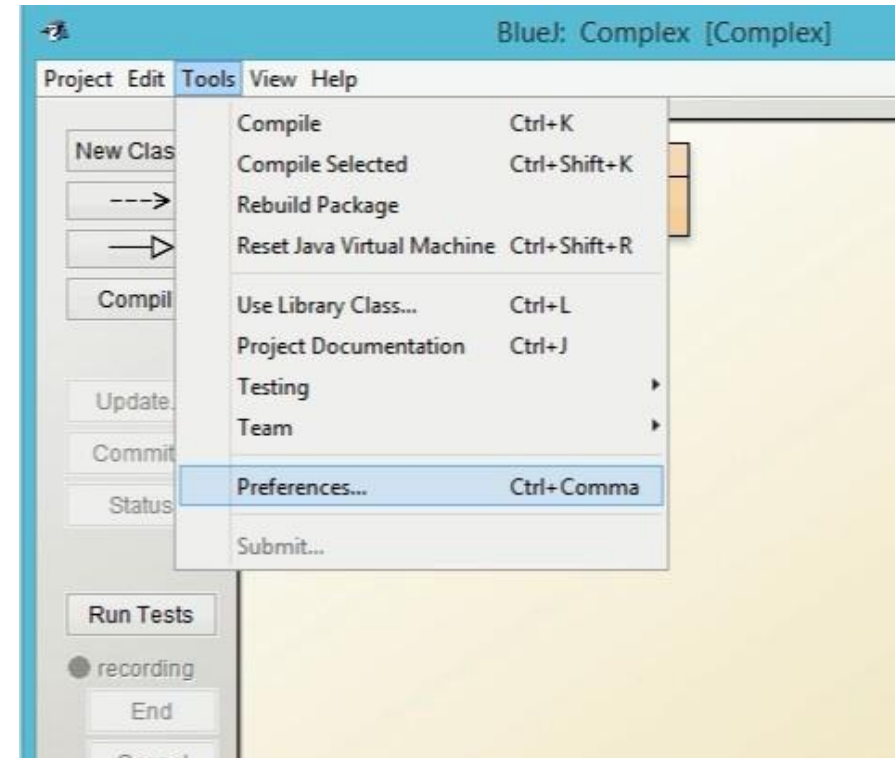
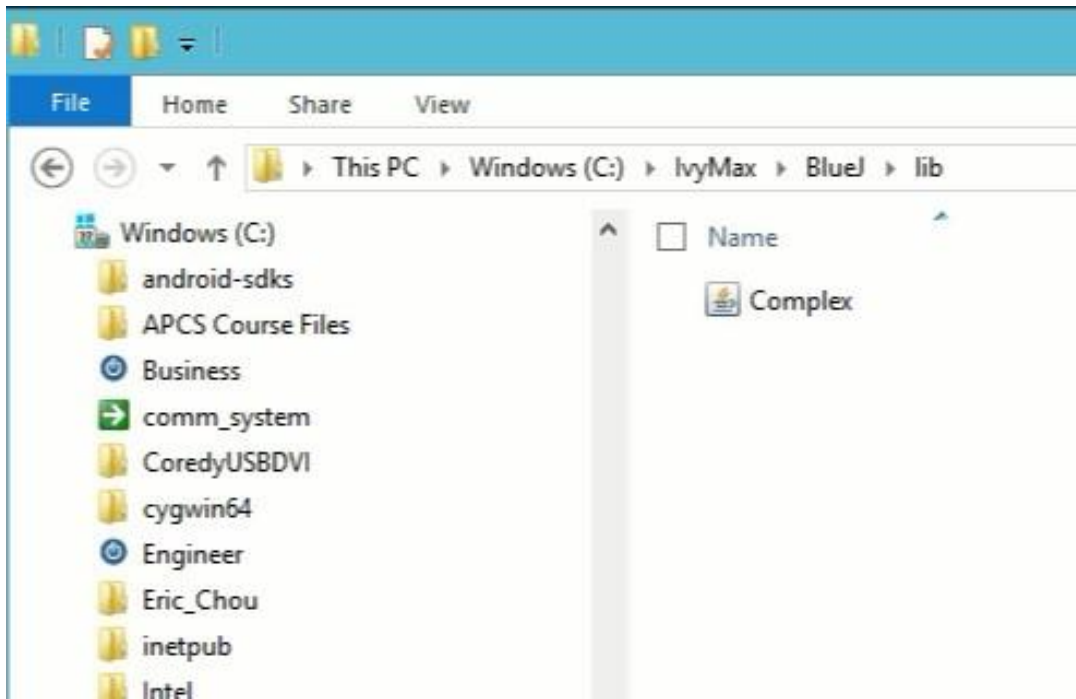


# Locate Complex.jar in the project directory



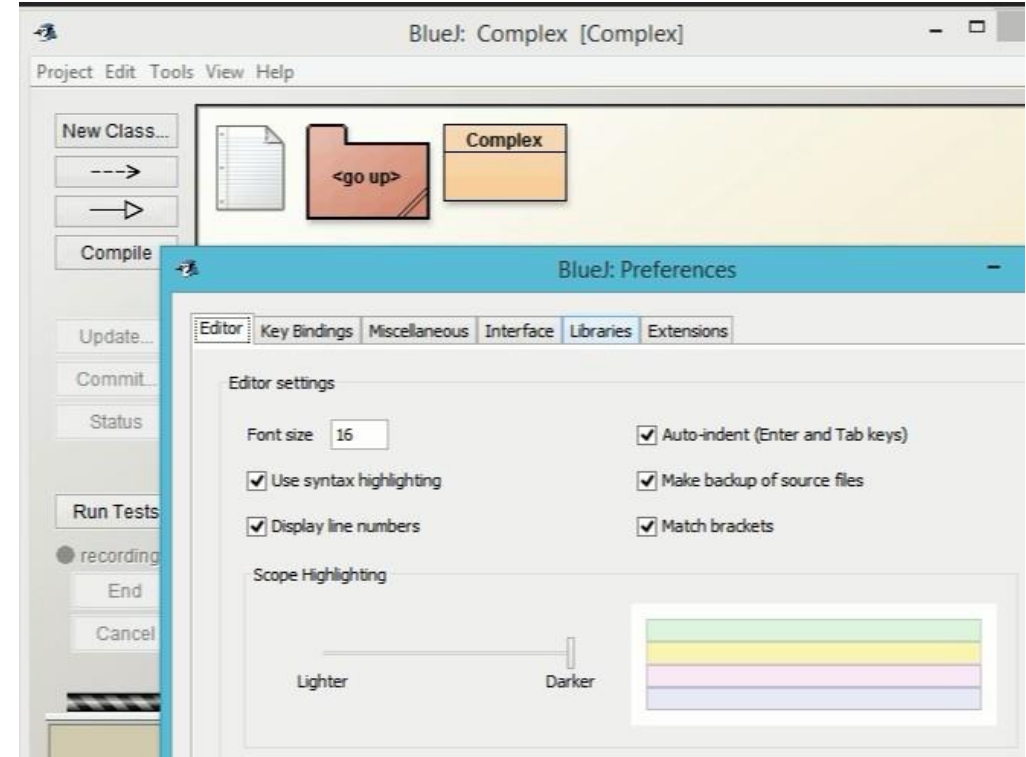
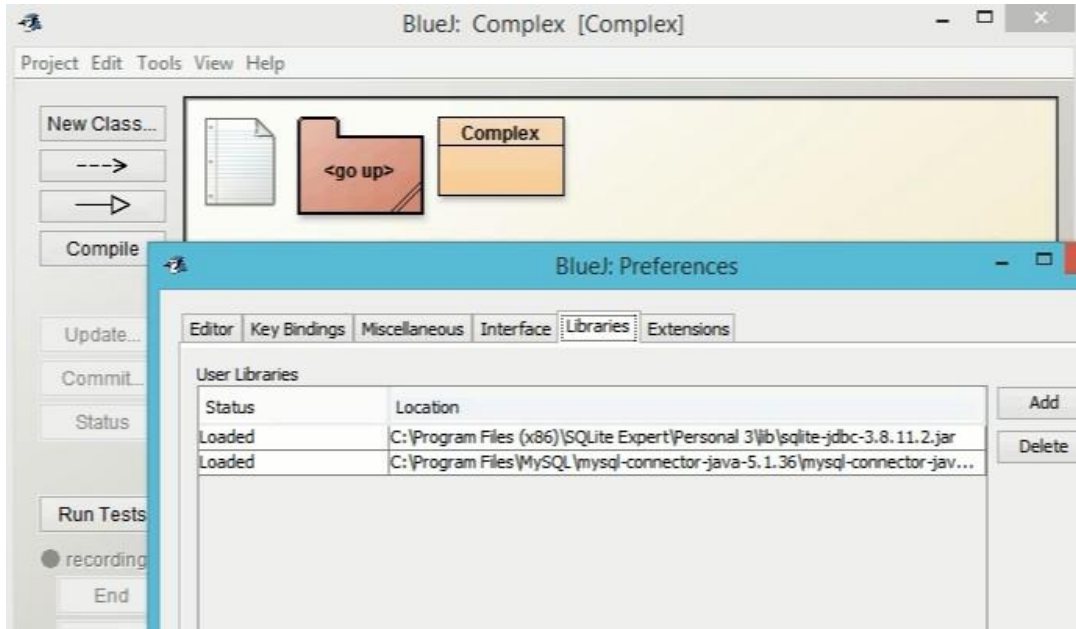


# Cut the Complex.jar File in project directory

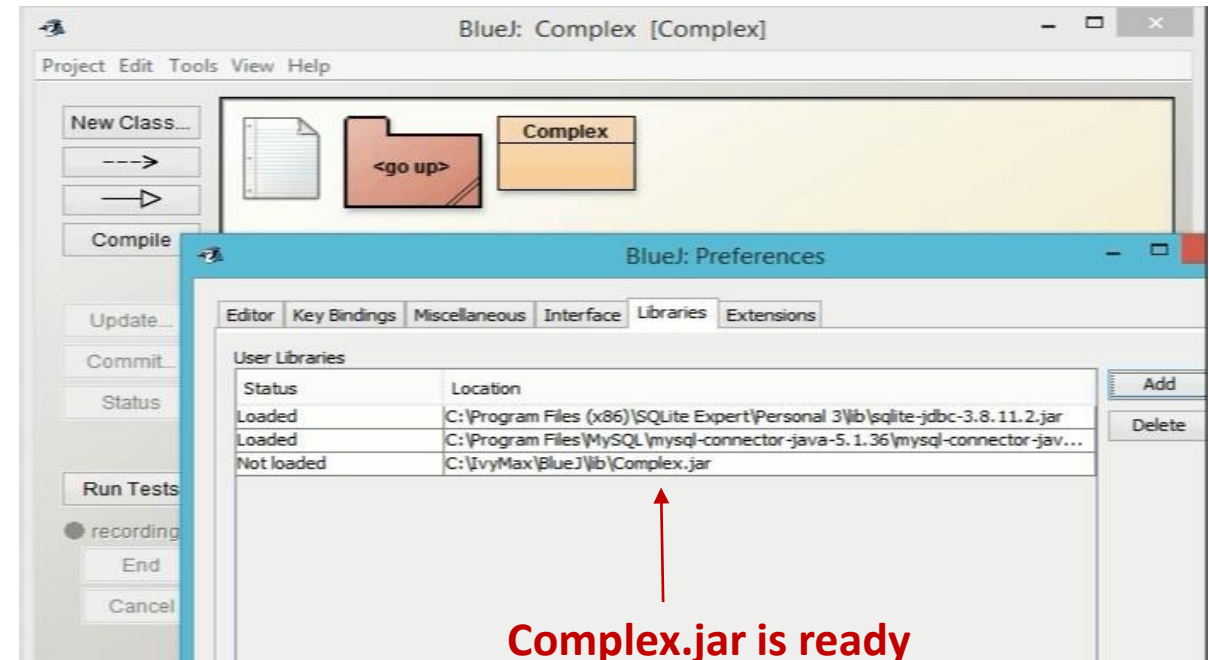
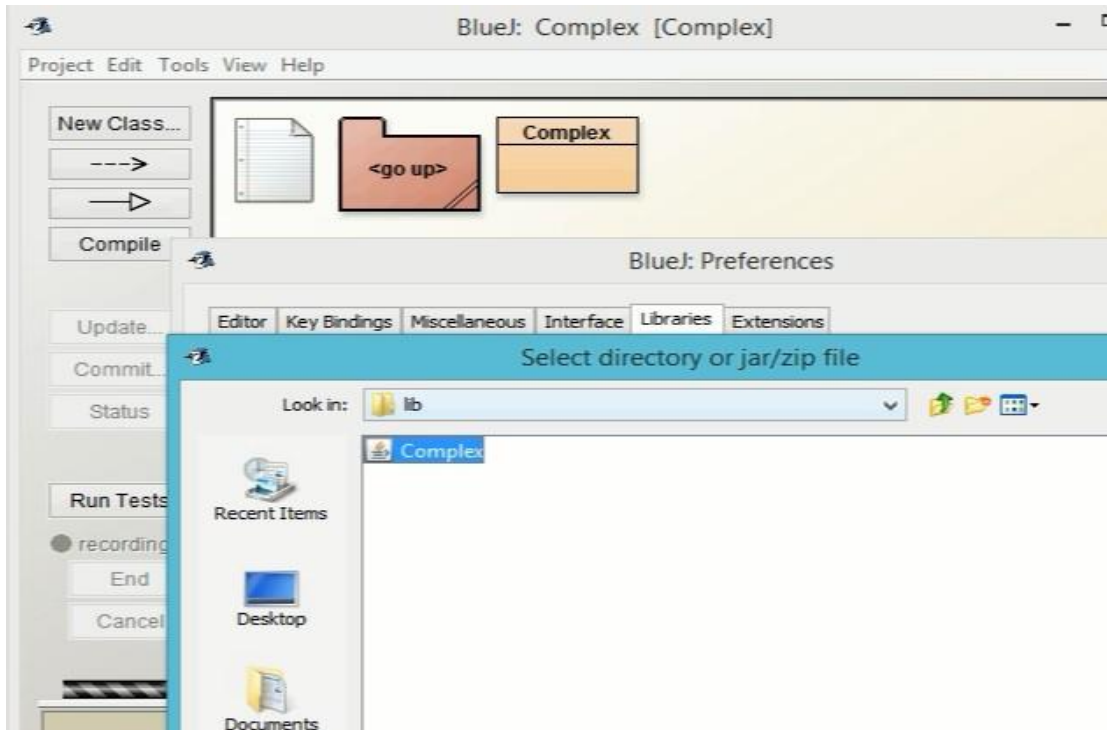


Put into the destination library directory  
Then, pick Tools-> Preferences

---



# Pick Libraries and add a User Library



Pick the Complex.jar file at the destination library directory. It will show up at the User Libraries Paths.

Complex.Complex	
double r; double i;	real part of the complex number imaginary part of the complex number
<b>Constructor Methods:</b> public Complex() public Complex(double rr) public Complex(double rr, double ii) <b>Getter Methods:</b> public double getR() public double getI() public int getQuadrant() public double getTheta() public double getThetaDegree()	Create a zero complex number 0 + 0 I Create a real number in complex format rr + 0 i Create a complex number rr + ii i  Get the real part of the complex number Get the imaginary part of the complex number Get the Quadrant of this complex number Get the angle of this complex number from x-axis (positive) Get the angle of this complex number in degrees

# UML for Complex.Complex

---

Complex.Complex	
<b>Setter Methods:</b> public void setR(double rr) public void setI(double ii) public void setComplex(double rr, double ii)	Set the real part of the complex number Set the imaginary part of the complex number Set the complex number
<b>String and Equals Methods:</b> public String toString() public String toString2() public String toString4() public boolean equals(Complex cc) public boolean equals4(Complex cc) public boolean equals8(Complex cc) public boolean notEquals(Complex cc)	Convert the complex number to string no formatting Convert the complex number to string with 2 decimals Convert the complex number to string with 4 decimals Check equality without formatting Check equality with significant 4 digits Check equality with significant 8 digits Check in-equality

# UML for Complex.Complex

---

---

## Complex.Complex

### Rotation Methods:

```
public Complex rotate(double theta1)
public Complex rotateDegree(double degree)
```

Rotate the complex number with an angle of theta1  
Rotate the complex number with an angle of “degree”

### Mathematical Methods:

```
public double abs()
public Complex add(Complex cc)
public Complex minus(Complex cc)
public Complex neg()
public Complex conjugate()
public Complex inverse()
public Complex multiplyR(double rr)
public Complex multiplyI(double ii)
public Complex multiply(Complex cc)
public Complex divideR(double rr)
public Complex divideI(double ii)
public Complex divide(Complex cc)
public Complex pow(double exp)
```

Get the absolute value.  
Add a complex number  
Minus a complex number  
Negate a complex number  
Find the conjugate of the complex number  
Find the inverse of the complex number  
Multiply the complex number with a real scalar.  
Multiply the complex number with a imaginary scalar.  
Multiply a complex number  
Divide the complex number with a real scalar.  
Divide the complex number with a imaginary scalar.  
Divide a complex number  
Find the complex number to the order of exp

---

# UML for Complex.Complex

---

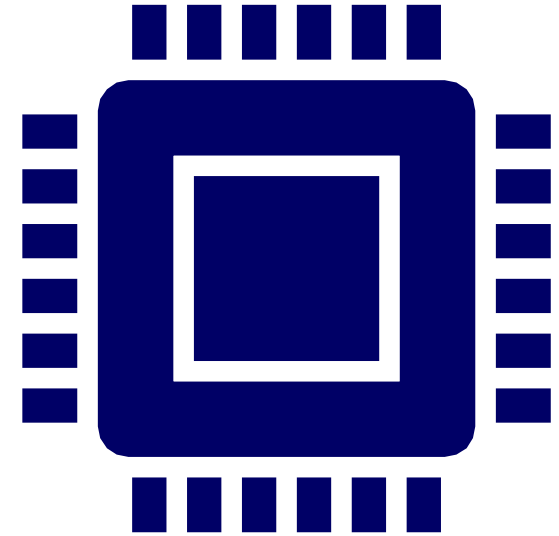


# Now, you are ready to use Complex.Complex Class

---

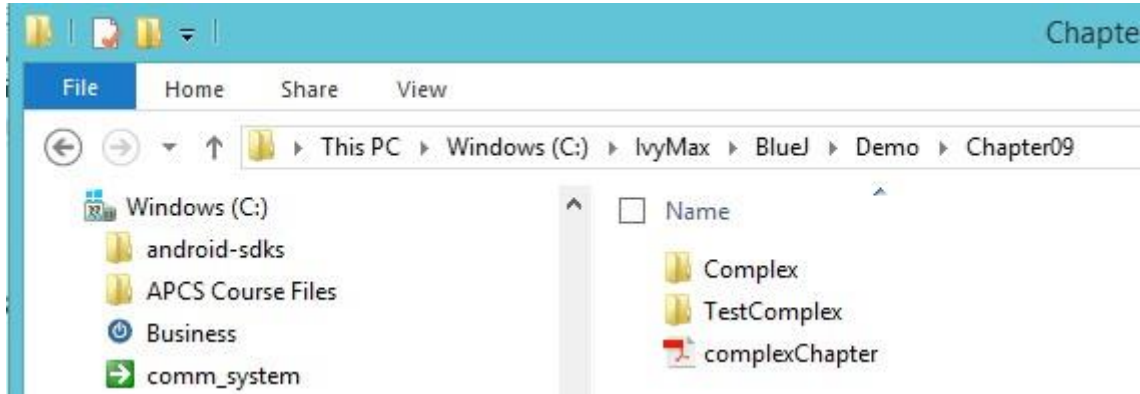
- Do programming as you usually do, put a import statement like:

```
package Complex.Complex;  
at the beginning of your  
program.
```

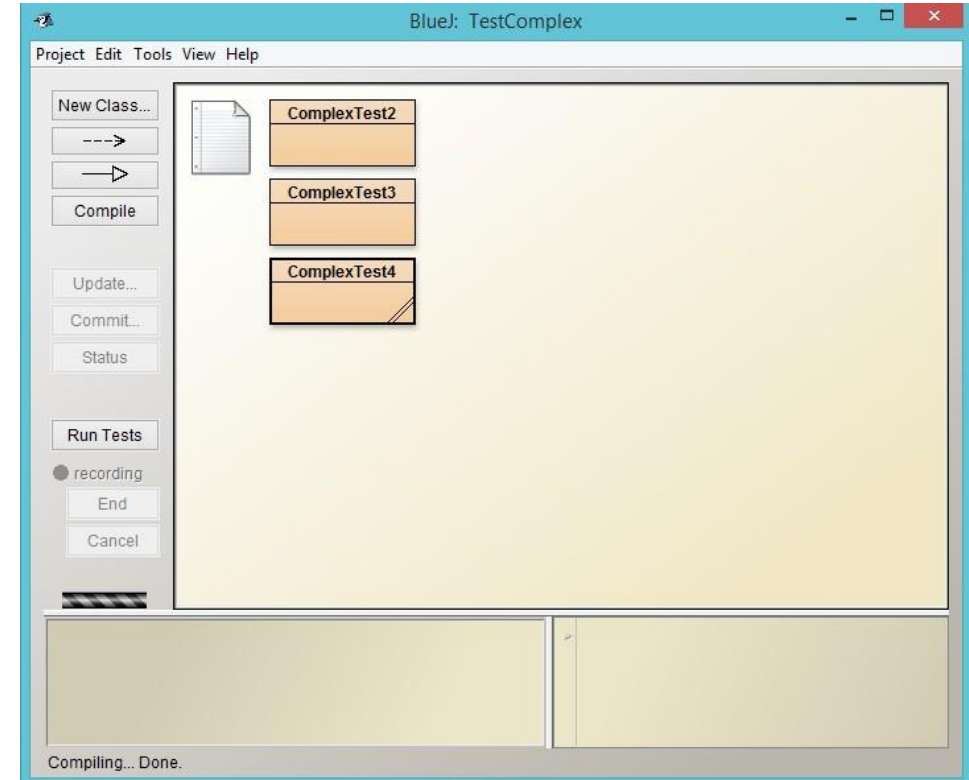




## Create a new project named TestComplex

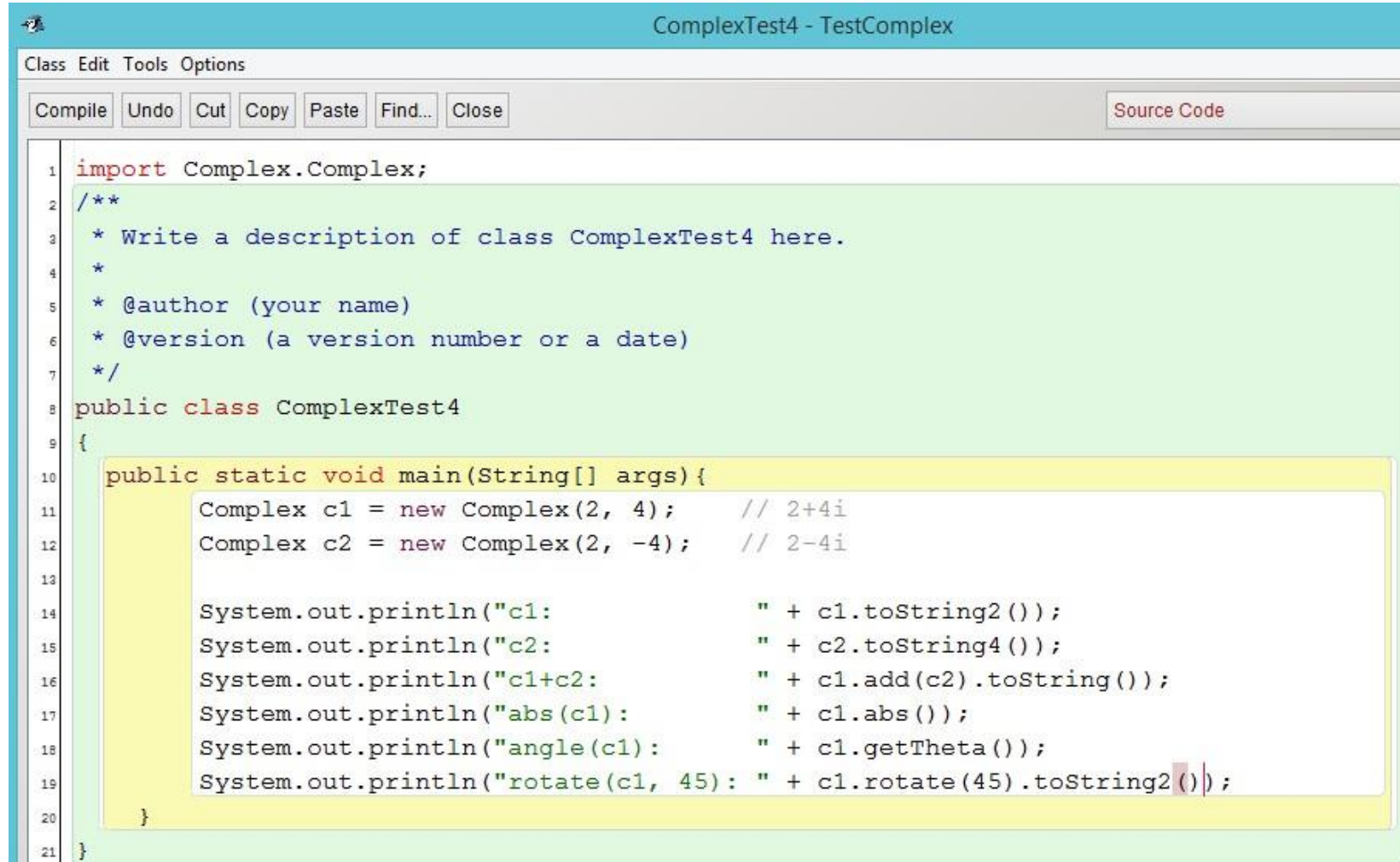


New Project **TextComplex** in BlueJ  
New Class **ComplexTest4** (or something like it)

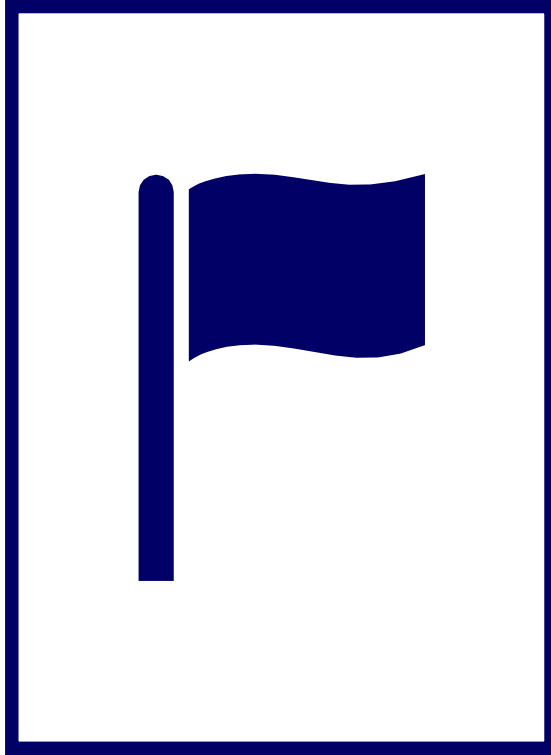
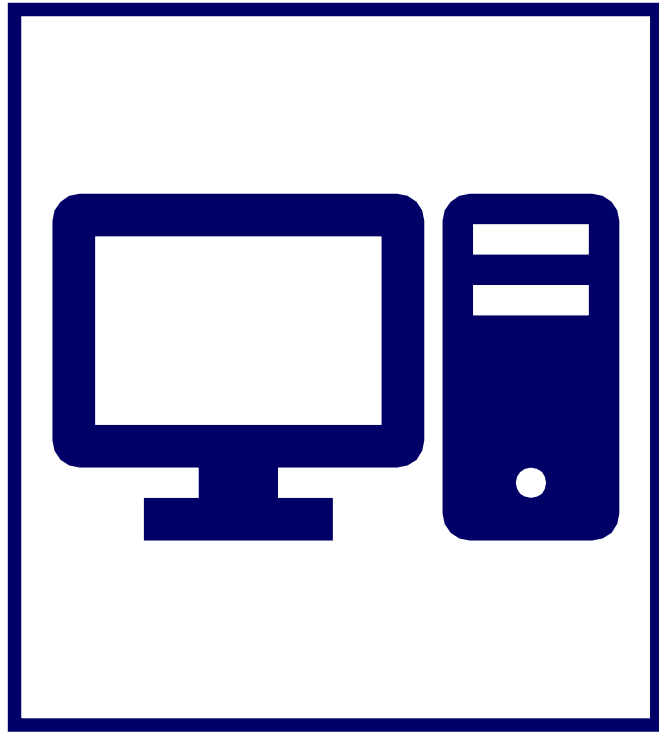


Now, try to create your own program to test it.

Create a  
program like  
this and test it.



```
ComplexTest4 - TestComplex
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Close Source Code
1 import Complex.Complex;
2 /**
3  * Write a description of class ComplexTest4 here.
4  *
5  * @author (your name)
6  * @version (a version number or a date)
7  */
8 public class ComplexTest4
9 {
10     public static void main(String[] args){
11         Complex c1 = new Complex(2, 4);    // 2+4i
12         Complex c2 = new Complex(2, -4);   // 2-4i
13
14         System.out.println("c1:           " + c1.toString2());
15         System.out.println("c2:           " + c2.toString4());
16         System.out.println("c1+c2:       " + c1.add(c2).toString());
17         System.out.println("abs(c1):     " + c1.abs());
18         System.out.println("angle(c1):   " + c1.getTheta());
19         System.out.println("rotate(c1, 45): " + c1.rotate(45).toString2());
20     }
21 }
```



## Project: Building a Complex Package

---

Student should work on this project in Class, or work on this project as a homework.