AP Computer Science A

Java Programming Essentials [Ver.4.0]

Unit 3: Class Creation

CHAPTER 11: CLASSES AND OBJECTS

DR. ERIC CHOU
IEEE SENIOR MEMBER

AP Computer Science Curriculum

- Abstraction and Program Design (T 3.1)
- Impact of Program Design (T 3.2)
- Anatomy of a Class (T 3.3)
- Constructors (T 3.4)
- Instance method: How to Write Them (T3.5)

Objectives:

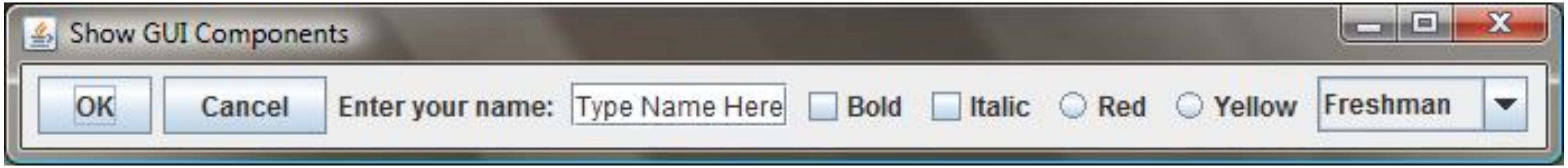
- Creation of Classes and Objects
- Static Programs
- Object-Oriented programs
- Instance Variables
- Instance Methods: How to Write Them
- Types of Classes
- Data Carriers (Array of Objects and Object of Arrays)
- Visibility and Scope

Class Definition and Object Creation

Lecture 1

Introduction to Object-Oriented Programming

- After learning the preceding chapters, you are capable of solving many programming problems using selections, loops, methods, and arrays. However, these Java features are not sufficient for developing graphical user interfaces and large scale software systems. Suppose you want to develop a graphical user interface as shown below. How do you program it?
- Without Object-Oriented Programming, things shall still work. Why we need Object-Oriented Programming?



Motivation for Object-Oriented Programming

- More Compatible for Event-Driven Programming
- More Manageable for GUI Components
- More Organized Data and Methods related to a Certain Objects
- **Replacing:**
- (1) Library
- (2) Data Records
- (3) Event-Loop (Execution Flow)
- (4) Thread Execution Control

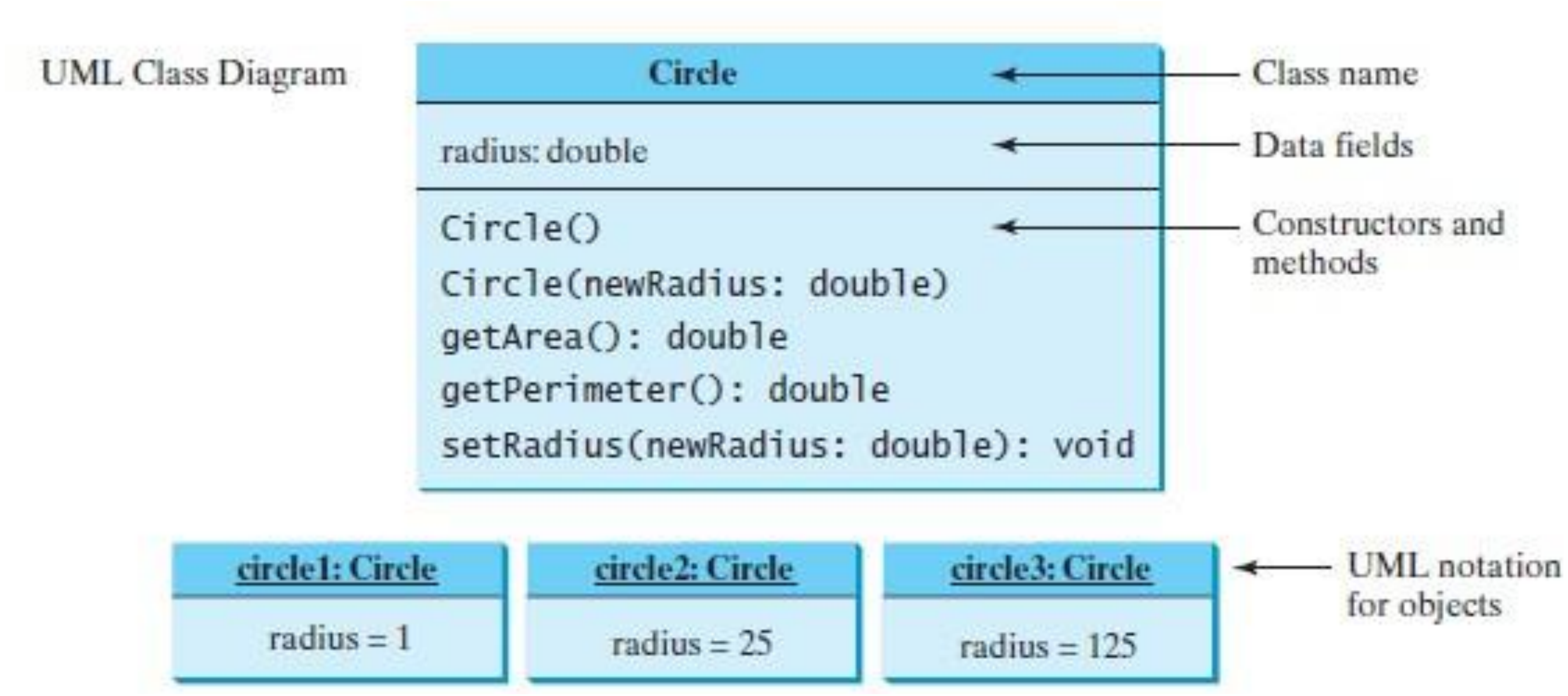
OO Programming Concepts

- Object-oriented programming (OOP) involves programming using objects. An **object** represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
- An object has a unique identity, state, and behaviors. The *state* of an object consists of a set of **data fields** (also known as **properties**) with their current values. The *behavior* of an object is defined by a set of methods.

Why object is different from data?

- An object has both a **state** and **behavior**. The state defines the object, and the behavior defines what the object does.

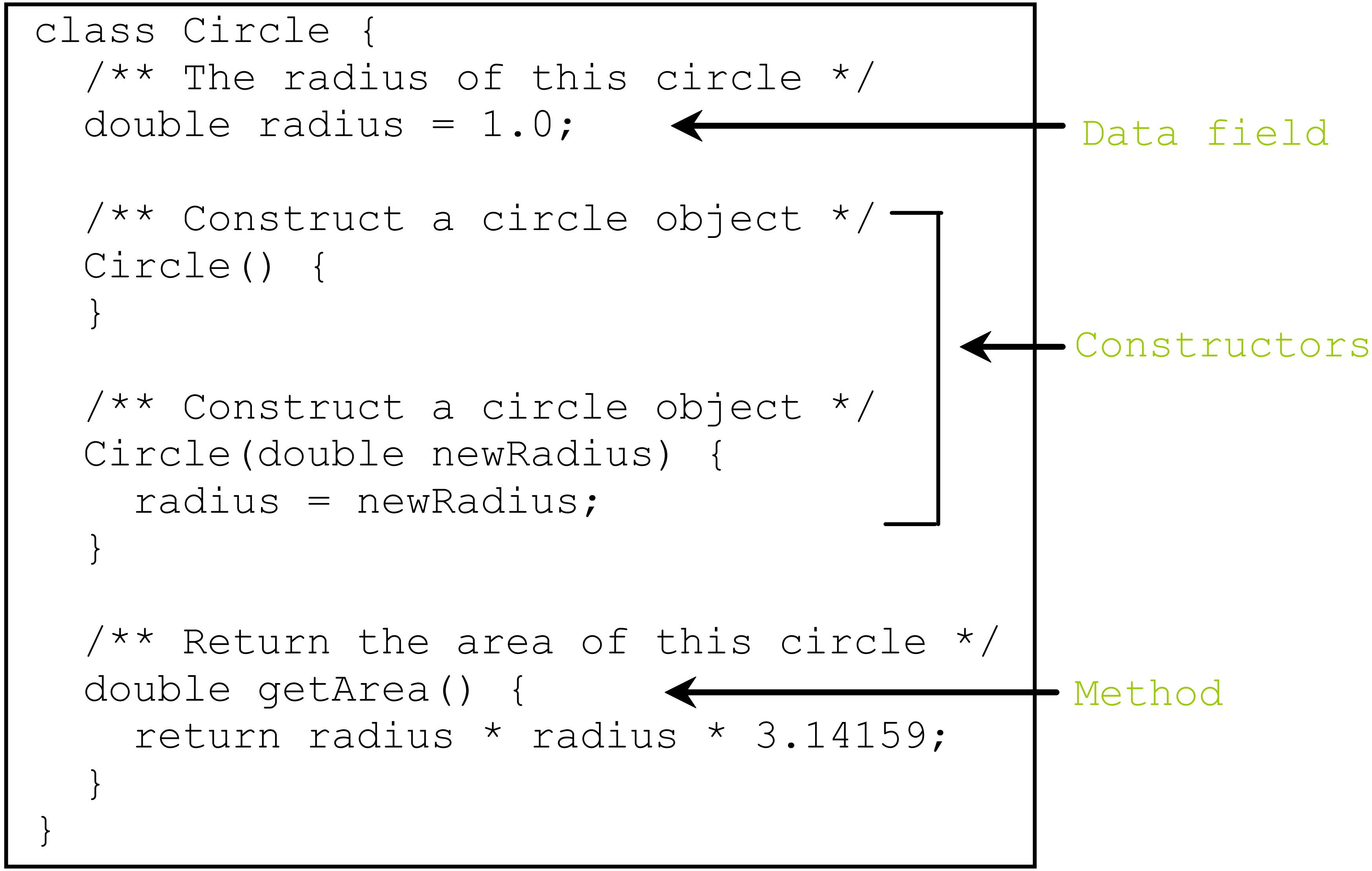
Objects: UML Class/Object Diagram



Classes

Classes are constructs that define objects of the same type. A Java class uses variables to define data fields and methods to define behaviors. Additionally, a class provides a special type of methods, known as constructors, which are invoked to construct objects from the class.

Classes



```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

  /** Construct a circle object */
  Circle() {
  }

  /** Construct a circle object */
  Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * 3.14159;
  }
}
```

Demonstration Program

TESTSIMPLECIRCLE.JAVA

Constructors

Lecture 2

Constructors

- Constructors are a special kind of methods that are invoked to construct objects.

```
Circle() {
}
Circle(double newRadius) {
  radius = newRadius;
}
```

Constructors, cont.

A **constructor** with no parameters is referred to as a *no-arg constructor*. If no constructor is given, default one will be used.

- Constructors must have the same name as the class itself.
- Constructors do not have a return type—not even void.
- Constructors are invoked using the new operator when an object is created. Constructors play the role of initializing objects.

Creating Objects Using Constructors

```
new ClassName();
```

Example:

```
new Circle();
new Circle(5.0);
```

Declaring Object Reference Variables

- To reference an object, assign the object to a reference variable. For an object, you may have as many reference variable (pointer) as you wish.
- To declare a reference variable, use the syntax:

```
ClassName objectRefVar;
```

Example:

```
Circle myCircle;
```

Declaring/Creating Objects in a Single Step

```
ClassName objectRefVar = new ClassName();
```

Example:

```
Circle myCircle = new Circle();
```

Accessing Object's Members

(Properties or Methods)

Referencing the object's data:

```
objectRefVar.data
```

e.g., `myCircle.radius`

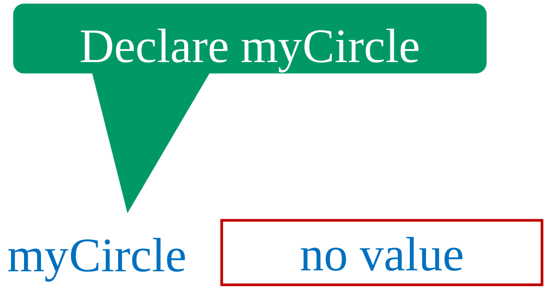
Invoking the object's method:

```
objectRefVar.methodName(arguments)
```

e.g., `myCircle.getArea()`

Trace Code

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



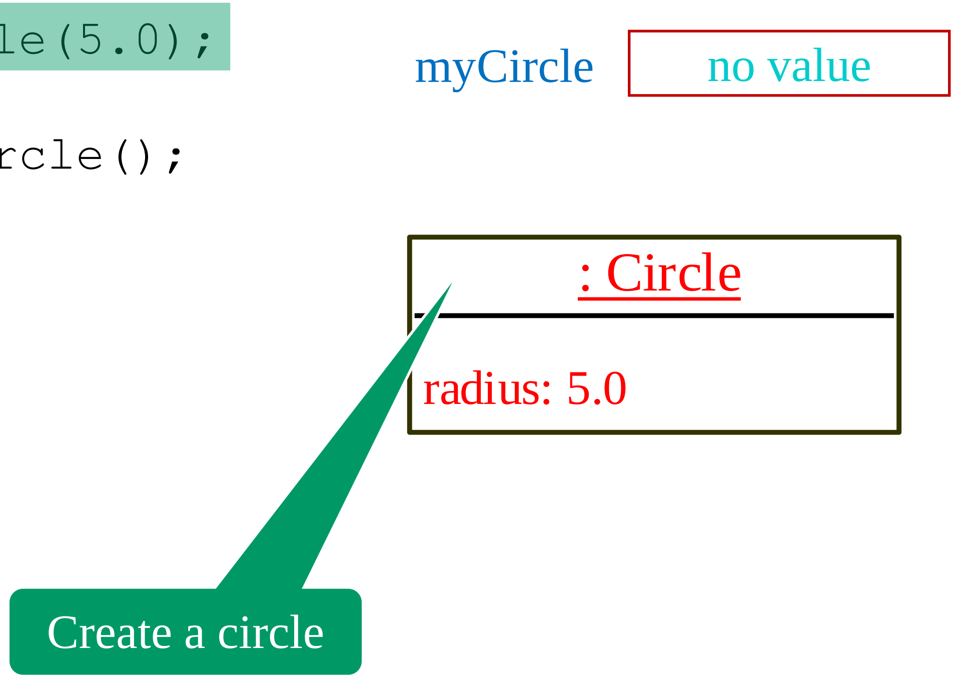
Declare myCircle

myCircle

no value

Trace Code

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



myCircle

no value

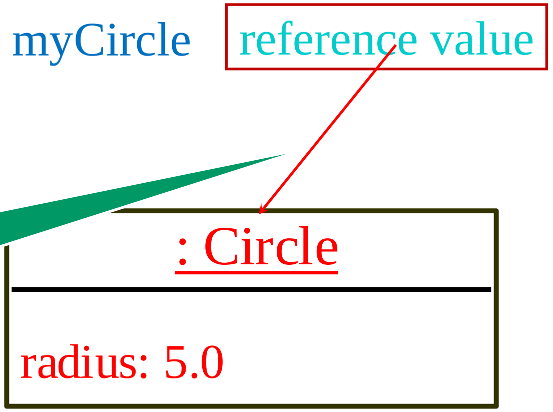
: Circle

radius: 5.0

Create a circle

Trace Code

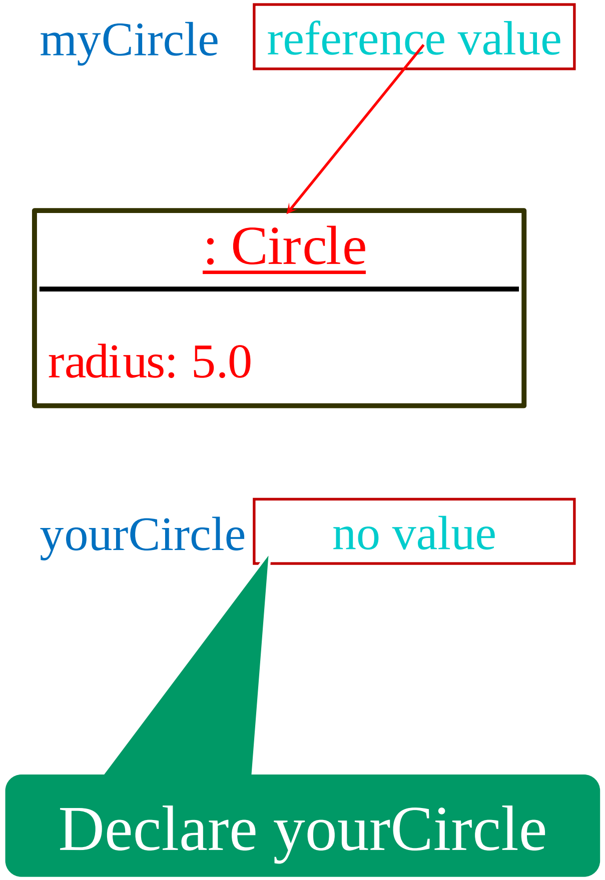
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Assign object reference to myCircle

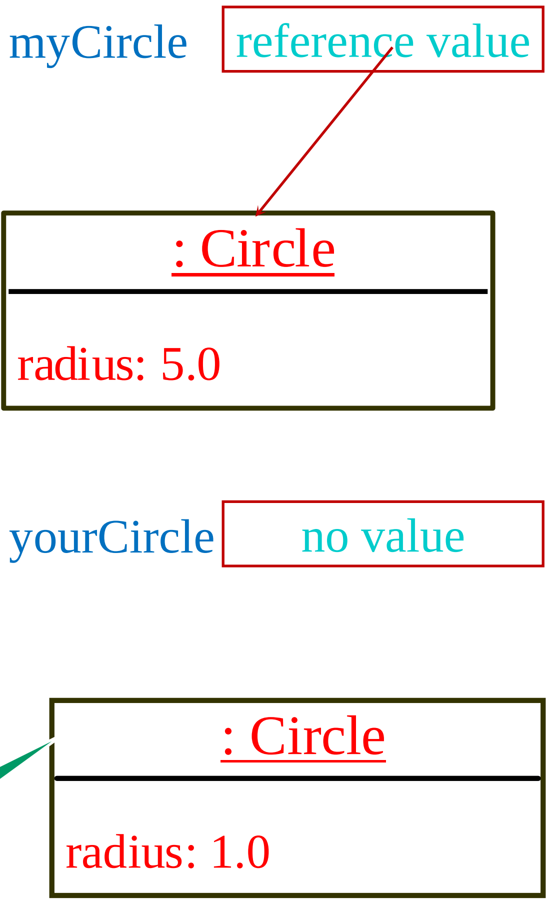
Trace Code

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Trace Code

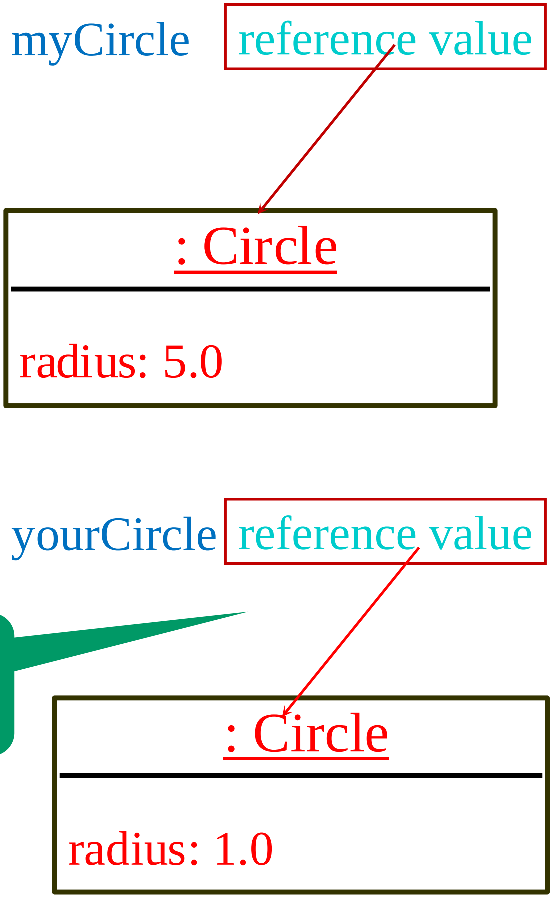
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Create a new Circle object

Trace Code

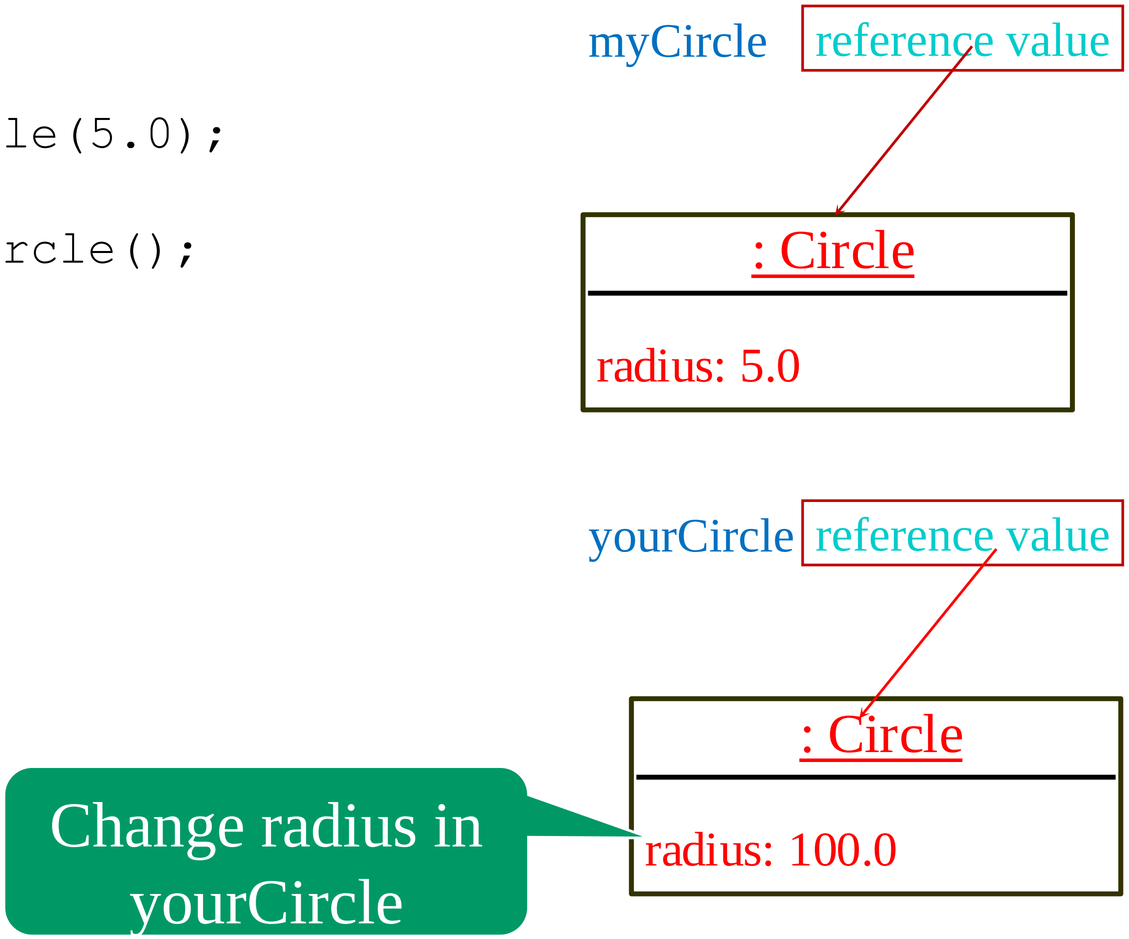
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Assign object reference to yourCircle

Trace Code

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Change radius in yourCircle

Caution

(static members belong to a class, non-static members belongs to objects)

- Recall that you use

```
   Math.methodName(arguments)  (e.g.,  Math.pow(3,  2.5))
```

- to invoke a method in the Math class. Can you invoke getArea() using **Circle**.getArea()? The answer is no. All the methods used before this chapter are static methods, which are defined using the static keyword. However, getArea() is non-static. It must be invoked from an object using

```
   objectRefVar.methodName(arguments)

       (e.g.,  myCircle.getArea()).
```

- More explanations will be given in the section on “Static Variables, Constants, and Methods.”

Overview of Class and Objects from Program Structure Point of View

Lecture 3

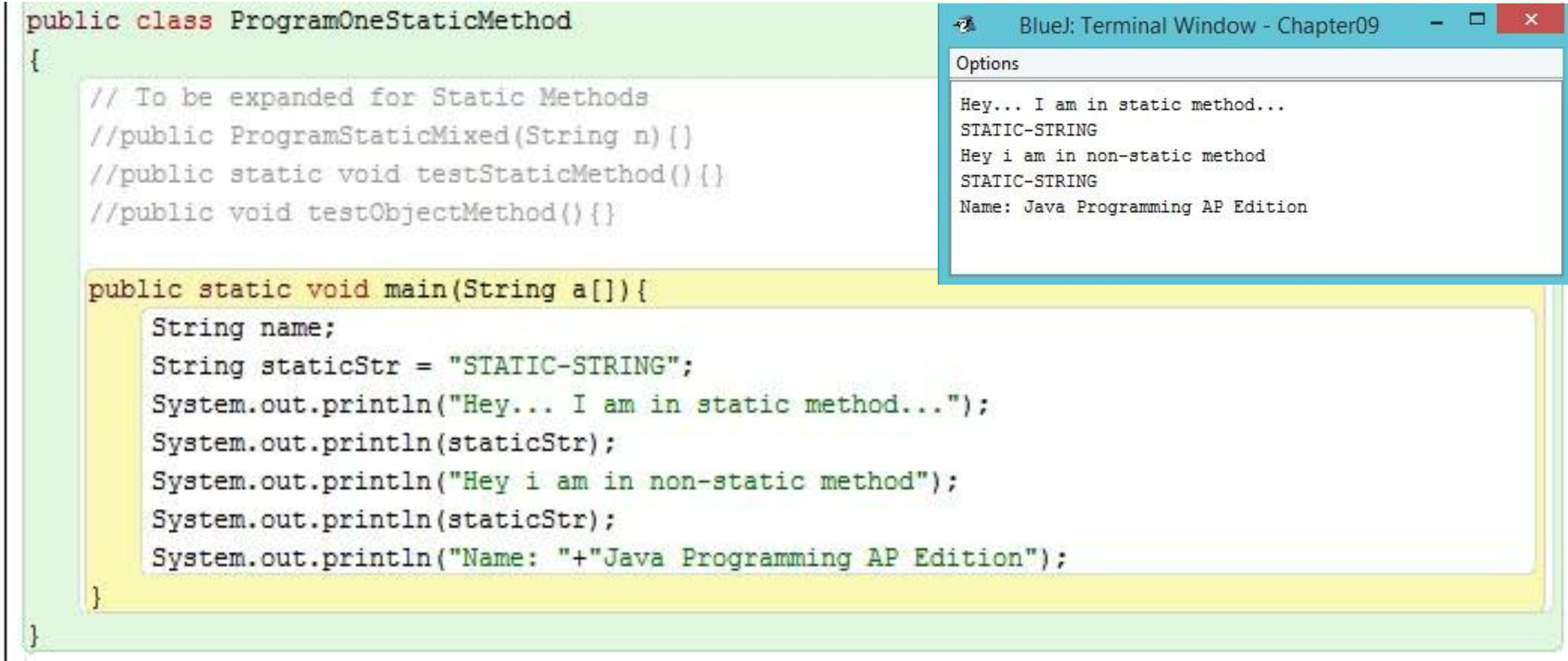
One Static Method Program

ProgramOneStaticMethod.java

- One Method: public static void main(String[] args) {…}
- All variable defined at the beginning of the main method.
- Simple program. No calling other methods. (Elementary Programming: Sequential Programming)

One Static Method Program

ProgramOneStaticMethod.java



```
public class ProgramOneStaticMethod
{
    // To be expanded for Static Methods
    //public ProgramStaticMixed(String n){}
    //public static void testStaticMethod(){}
    //public void testObjectMethod(){}

    public static void main(String a[]){
        String name;
        String staticStr = "STATIC-STRING";
        System.out.println("Hey... I am in static method...");
        System.out.println(staticStr);
        System.out.println("Hey i am in non-static method");
        System.out.println(staticStr);
        System.out.println("Name: "+"Java Programming AP Edition");
    }
}
```

```
Hey... I am in static method...
STATIC-STRING
Hey i am in non-static method
STATIC-STRING
Name: Java Programming AP Edition
```

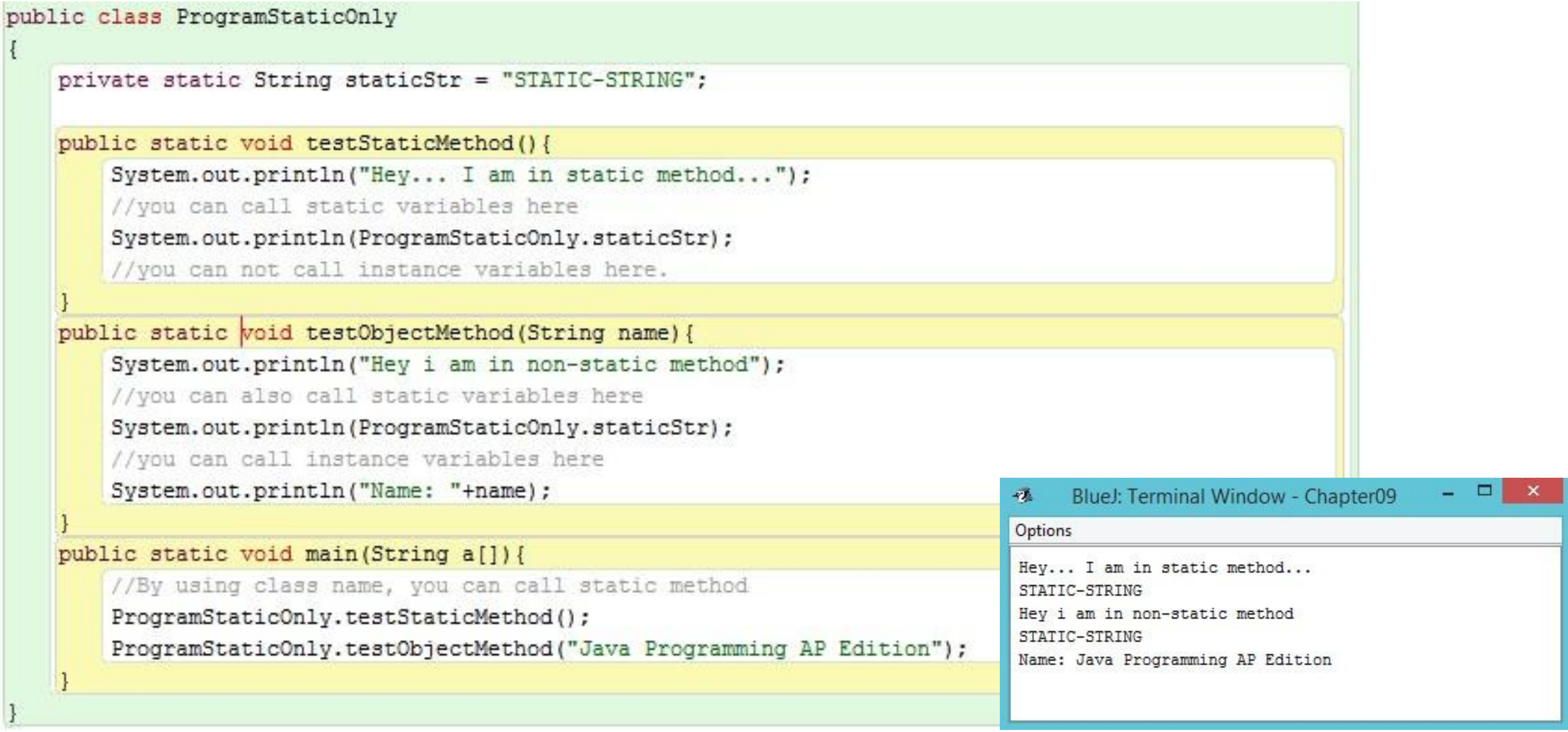
Static Method Only Program Structures

(ProgramStaticOnly.java Equivalent to Structural Programming)

- Global variables defined at the class properties declaration region as static variables.
- Local variables defined at the main method or other methods (or code blocks)
- Use ClassName.method() to call static methods or Use method() to call the methods (if only one class).
- Multiple class, you can only use ClassName.method() to call the method (similar to external functional call for structural programming language like C.)
- Equivalent to Structural Programming like C-language.

Static Method Only Program Structures

(ProgramStaticOnly.java Equivalent to Structural Programming)



```
public class ProgramStaticOnly
{
    private static String staticStr = "STATIC-STRING";

    public static void testStaticMethod(){
        System.out.println("Hey... I am in static method...");
        //you can call static variables here
        System.out.println(ProgramStaticOnly.staticStr);
        //you can not call instance variables here.
    }
    public static void testObjectMethod(String name){
        System.out.println("Hey i am in non-static method");
        //you can also call static variables here
        System.out.println(ProgramStaticOnly.staticStr);
        //you can call instance variables here
        System.out.println("Name: "+name);
    }
    public static void main(String a[]){
        //By using class name, you can call static method
        ProgramStaticOnly.testStaticMethod();
        ProgramStaticOnly.testObjectMethod("Java Programming AP Edition");
    }
}
```

```
Hey... I am in static method...
STATIC-STRING
Hey i am in non-static method
STATIC-STRING
Name: Java Programming AP Edition
```

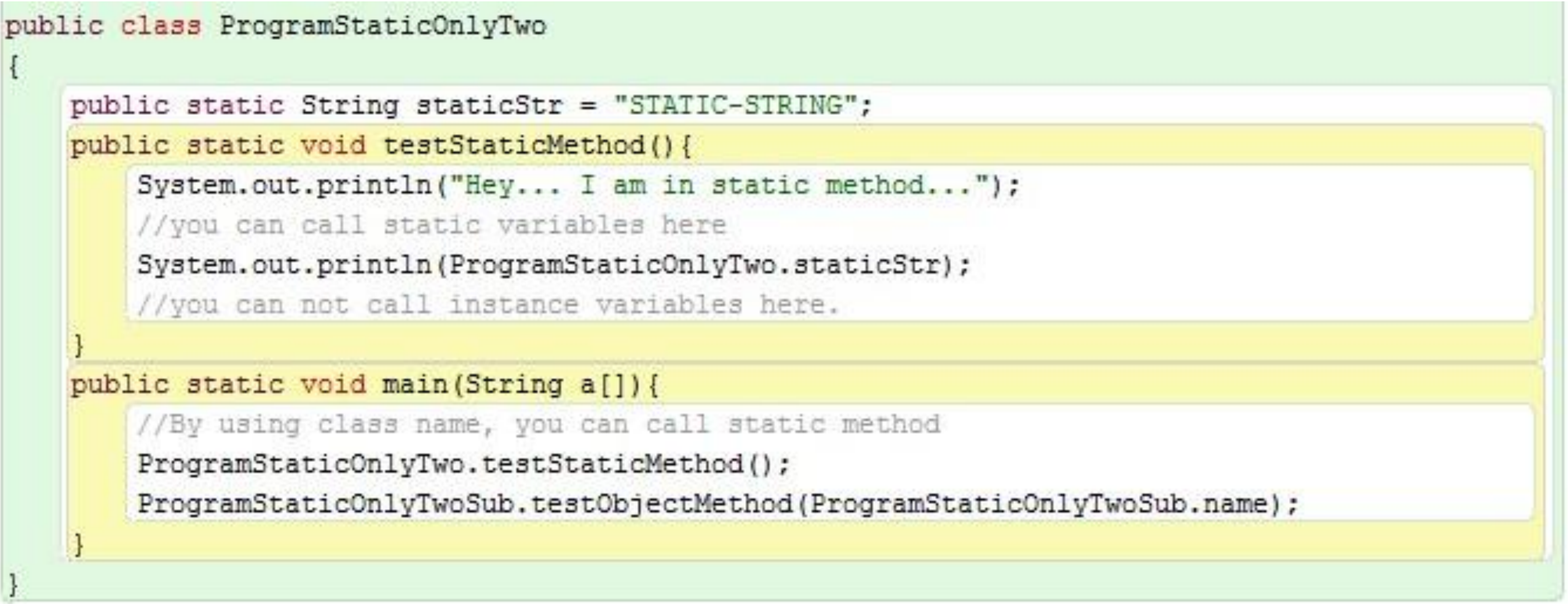
Static Method Only with Two Classes

(ProgramStaticOnlyTwo.java Equivalent to Structural Programming)

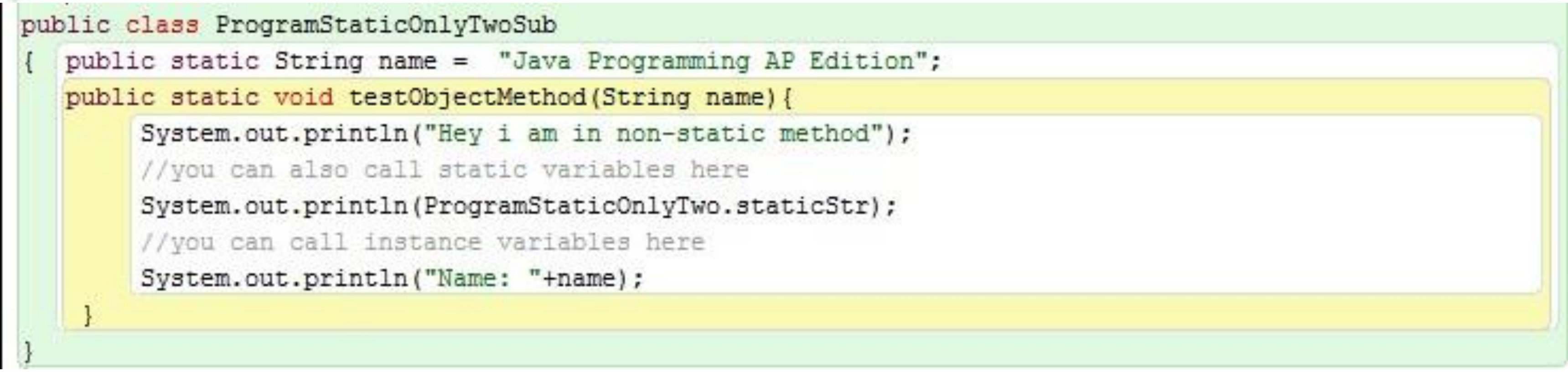
- Similar to Static Method Only Program except that two classes are used. (One is considered as an external program call.
- Because the two classes are in the same package, there is no need to import. If they are in different package, then the visibility modifiers need to be specified as public.
- Similar to external functional call in C-language.
- Math.random() call is considered one of such method calls.
- Two classes are mutually dependent (closely coupled). If they are independent, we call it uncoupled. (closely coupled programs are not considered to be safe.)

Static Method Only with Two Classes

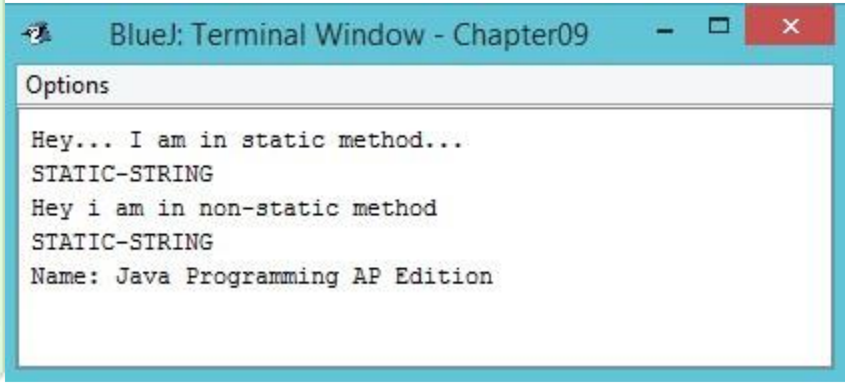
(ProgramStaticOnlyTwo.java Equivalent to Structural Programming)



```java
public class ProgramStaticOnlyTwo
{
    public static String staticStr = "STATIC-STRING";
    public static void testStaticMethod(){
        System.out.println("Hey... I am in static method...");
        //you can call static variables here
        System.out.println(ProgramStaticOnlyTwo.staticStr);
        //you can not call instance variables here.
    }
    public static void main(String a[]){
        //By using class name, you can call static method
        ProgramStaticOnlyTwo.testStaticMethod();
        ProgramStaticOnlyTwoSub.testObjectMethod(ProgramStaticOnlyTwoSub.name);
    }
}
```



```java
public class ProgramStaticOnlyTwoSub
{  public static String name =  "Java Programming AP Edition";
   public static void testObjectMethod(String name){
        System.out.println("Hey i am in non-static method");
        //you can also call static variables here
        System.out.println(ProgramStaticOnlyTwo.staticStr);
        //you can call instance variables here
        System.out.println("Name: "+name);
    }
}
```



```
Hey... I am in static method...
STATIC-STRING
Hey i am in non-static method
STATIC-STRING
Name: Java Programming AP Edition
```

Program Structure Static Mixed Style

Program: Example for static variables and methods

In java, static belongs to class. You can create static variables and static methods. You can call these directly by using class name, without creating instance.

Java static variables:

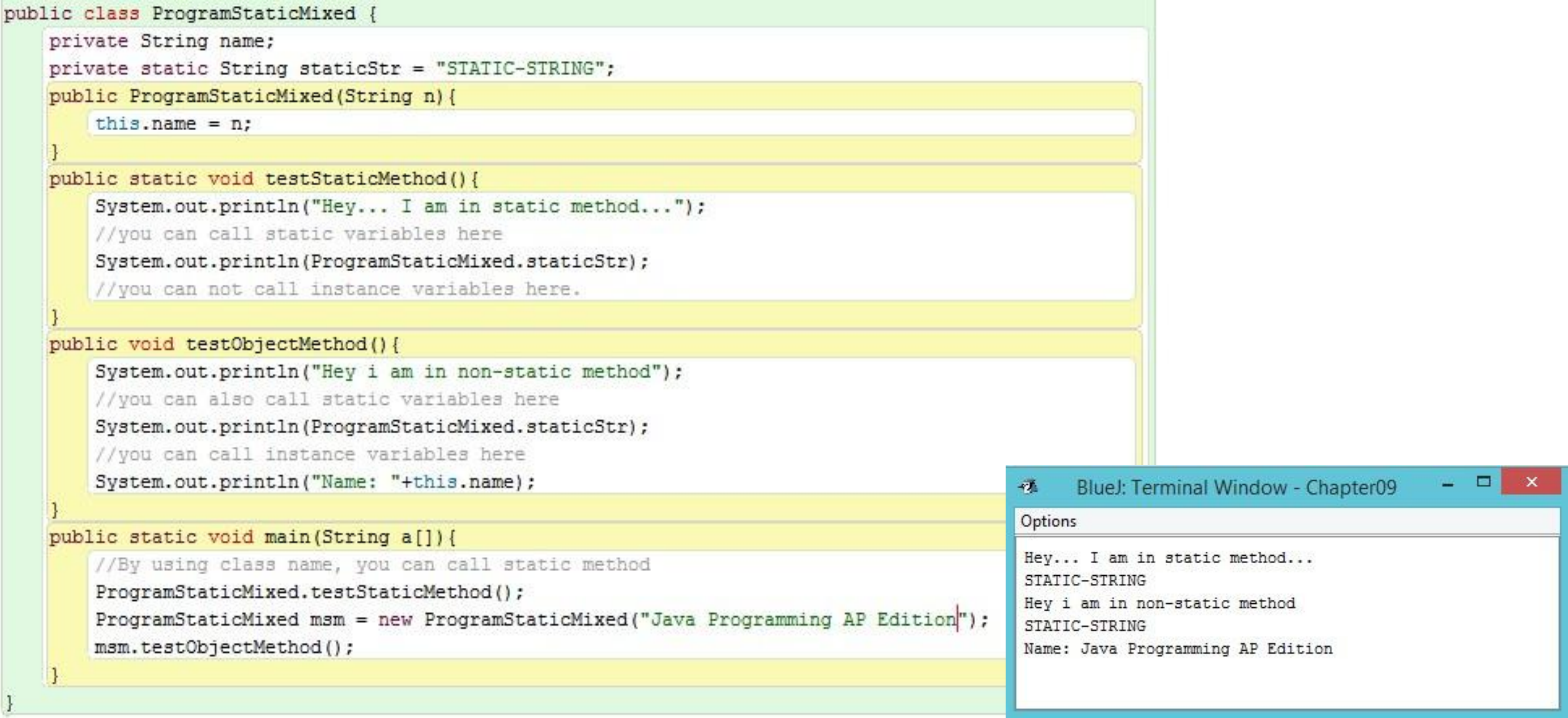
Static variables are belongs to the class and not to the object. These are only once, at the starting of the execution. Static variables are not part of object state, means there is only one copy of the values will be served to all instances. You can call static variable with reference to class name without creating an object. Below example shows how to create and call static variables.

Java static methods:

Static methods are also similar to static variables, you can access them with reference to class name, without creating object. Inside static methods, you cannot access instance variables or instance methods. You can only access static variables or static methods.

Program Structure Static Mixed Style

ProgramStaticMixed.java

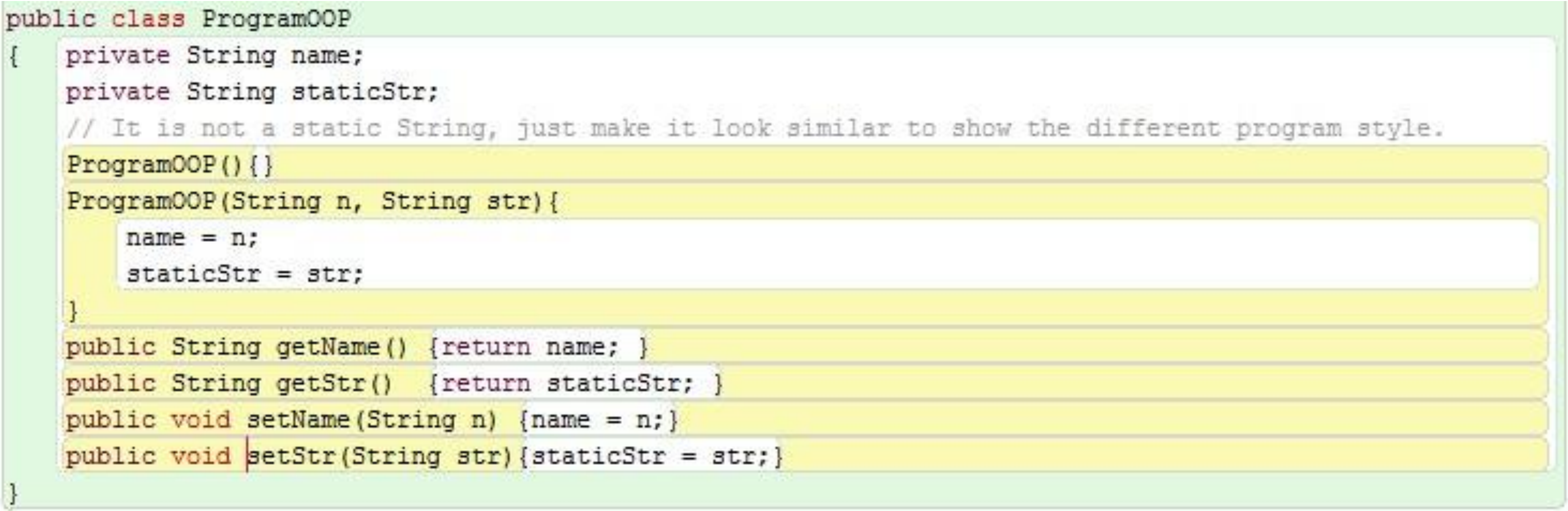


```java
public class ProgramStaticMixed {
    private String name;
    private static String staticStr = "STATIC-STRING";
    public ProgramStaticMixed(String n){
        this.name = n;
    }
    public static void testStaticMethod(){
        System.out.println("Hey... I am in static method...");
        //you can call static variables here
        System.out.println(ProgramStaticMixed.staticStr);
        //you can not call instance variables here.
    }
    public void testObjectMethod(){
        System.out.println("Hey i am in non-static method");
        //you can also call static variables here
        System.out.println(ProgramStaticMixed.staticStr);
        //you can call instance variables here
        System.out.println("Name: "+this.name);
    }
    public static void main(String a[]){
        //By using class name, you can call static method
        ProgramStaticMixed.testStaticMethod();
        ProgramStaticMixed msm = new ProgramStaticMixed("Java Programming AP Edition");
        msm.testObjectMethod();
    }
}
```

```
Hey... I am in static method...
STATIC-STRING
Hey i am in non-static method
STATIC-STRING
Name: Java Programming AP Edition
```

Data-Centric Program (OOP)

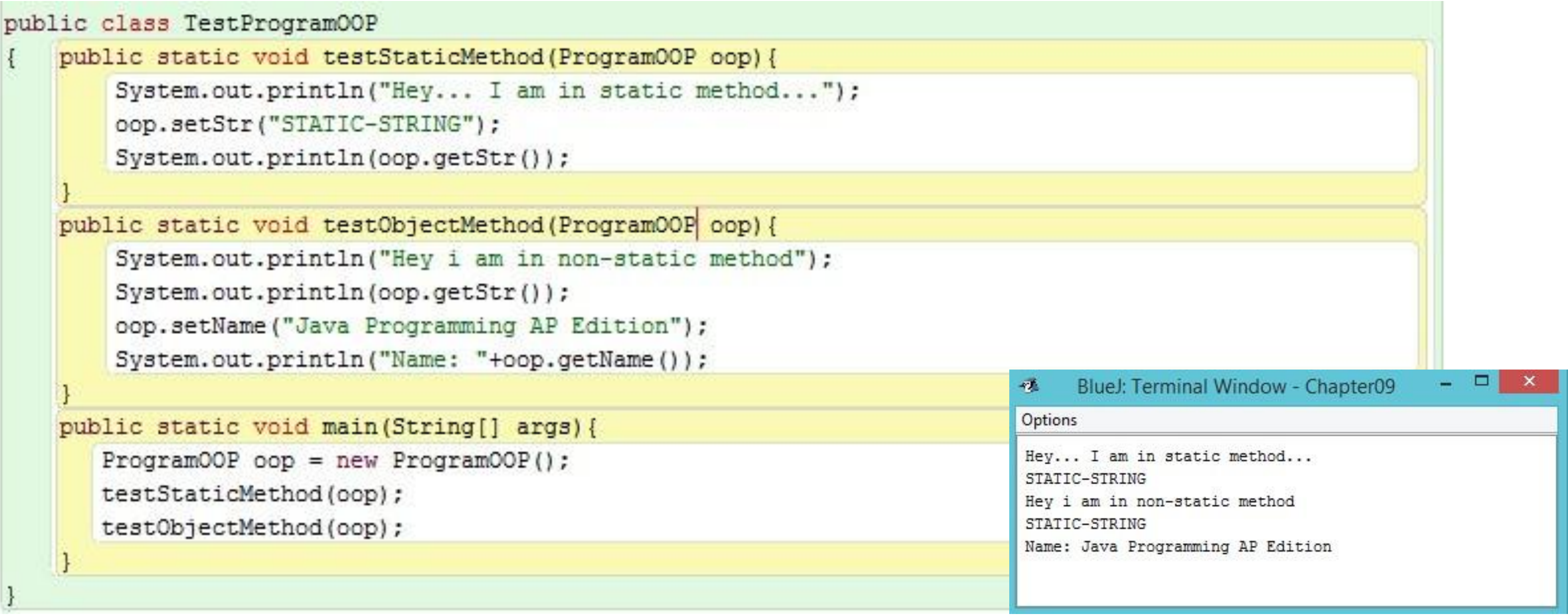
ProgramObjectOriented.java



```
public class ProgramOOP
{   private String name;
    private String staticStr;
    // It is not a static String, just make it look similar to show the different program style.
    ProgramOOP(){}
    ProgramOOP(String n, String str){
        name = n;
        staticStr = str;
    }
    public String getName()  {return name; }
    public String getStr()   {return staticStr; }
    public void setName(String n) {name = n;}
    public void setStr(String str){staticStr = str;}
}
```

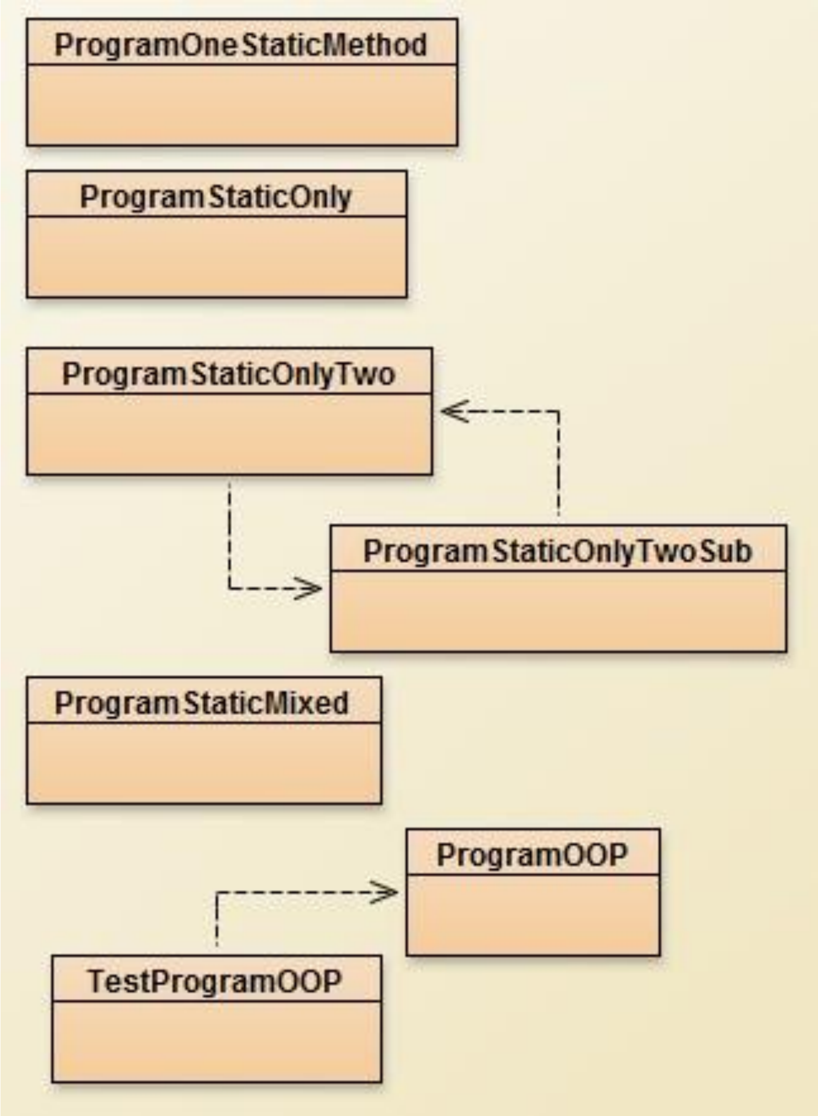
Tester Class for Data-Centric Program

TestObject-Oriented.java



```
public class TestProgramOOP
{   public static void testStaticMethod(ProgramOOP oop){
        System.out.println("Hey... I am in static method...");
        oop.setStr("STATIC-STRING");
        System.out.println(oop.getStr());
    }
    public static void testObjectMethod(ProgramOOP oop){
        System.out.println("Hey i am in non-static method");
        System.out.println(oop.getStr());
        oop.setName("Java Programming AP Edition");
        System.out.println("Name: "+oop.getName());
    }
    public static void main(String[] args){
       ProgramOOP oop = new ProgramOOP();
       testStaticMethod(oop);
       testObjectMethod(oop);
    }
}
```

```
Hey... I am in static method...
STATIC-STRING
Hey i am in non-static method
STATIC-STRING
Name: Java Programming AP Edition
```



Download the Zip file and Try on These Programs **(ClassAndObjectProgram.zip)**

- For the same method names, variable name, and the output, we have seen them in different program structures.

- In this lecture, I just try to give audience a picture of the many program structures that he might choose to use in Java. Some simple, some more complicated. The contents and the output really do not matter that much.

Demonstration Program

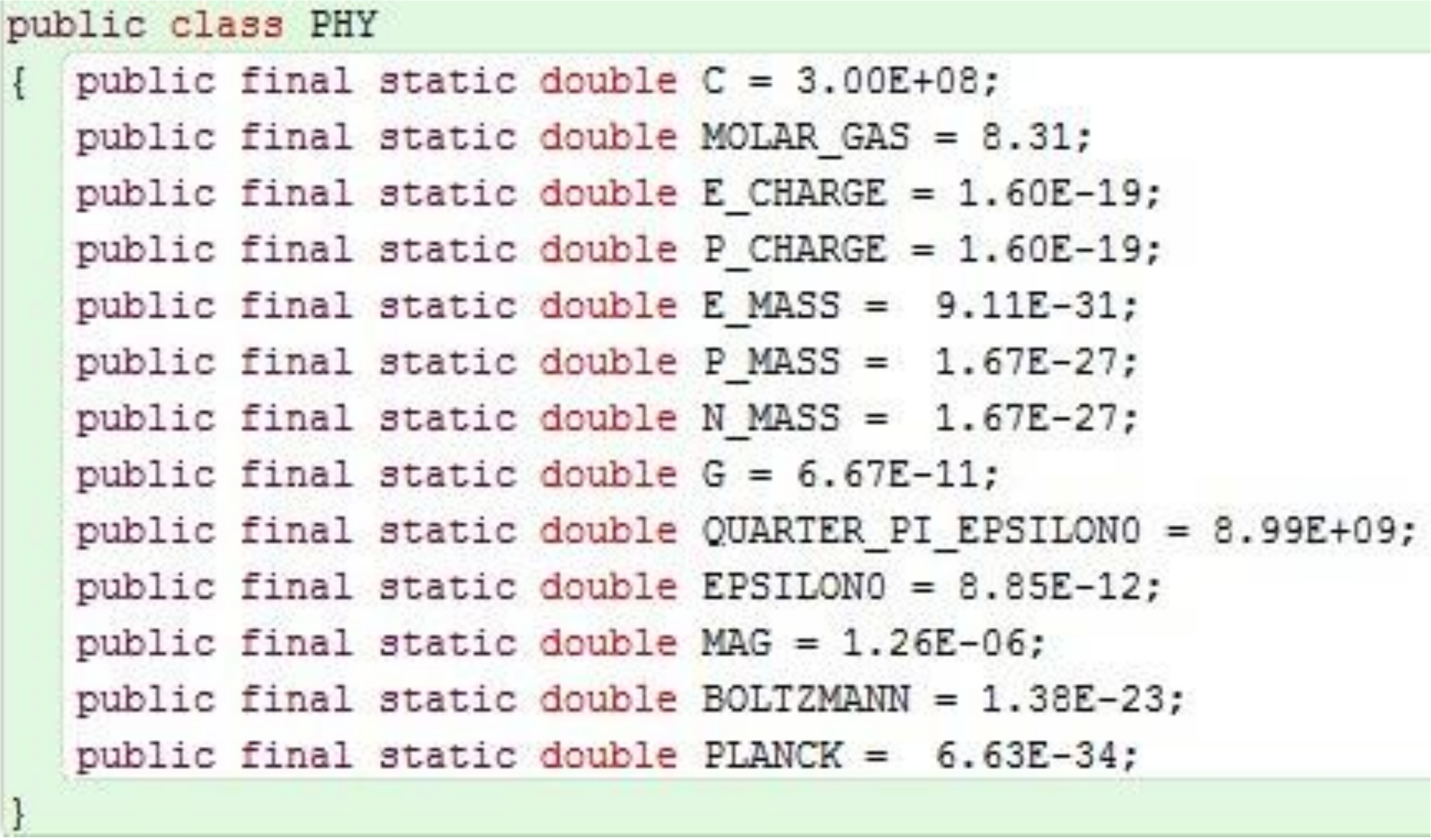
STATIC PROGRAM SERIES

Overview of Class and Objects from Data Structure Point of Views

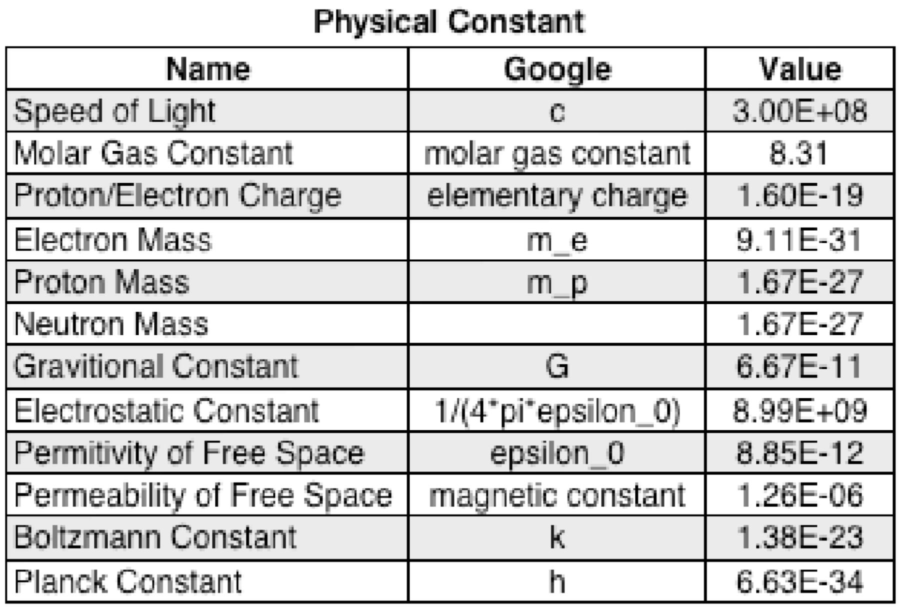
Lecture 4

Class as Collection of Constants

Class Variables, Constants



```
public class PHY
{  public final static double C = 3.00E+08;
   public final static double MOLAR_GAS = 8.31;
   public final static double E_CHARGE = 1.60E-19;
   public final static double P_CHARGE = 1.60E-19;
   public final static double E_MASS =  9.11E-31;
   public final static double P_MASS =  1.67E-27;
   public final static double N_MASS =  1.67E-27;
   public final static double G = 6.67E-11;
   public final static double QUARTER_PI_EPSILON0 = 8.99E+09;
   public final static double EPSILON0 = 8.85E-12;
   public final static double MAG = 1.26E-06;
   public final static double BOLTZMANN = 1.38E-23;
   public final static double PLANCK =  6.63E-34;
}
```



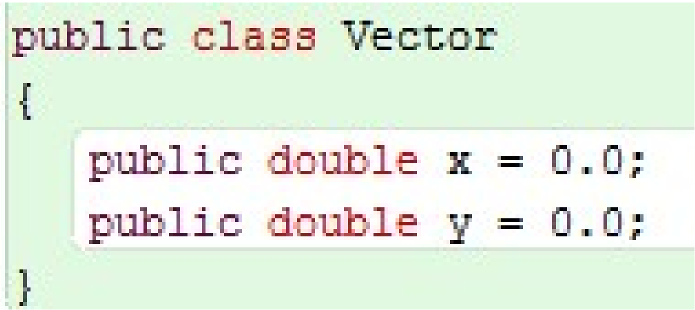
Physical Constant

Name	Google	Value
Speed of Light	c	3.00E+08
Molar Gas Constant	molar gas constant	8.31
Proton/Electron Charge	elementary charge	1.60E-19
Electron Mass	m_e	9.11E-31
Proton Mass	m_p	1.67E-27
Neutron Mass		1.67E-27
Gravitional Constant	G	6.67E-11
Electrostatic Constant	1/(4*pi*epsilon_0)	8.99E+09
Permitivity of Free Space	epsilon_0	8.85E-12
Permeability of Free Space	magnetic constant	1.26E-06
Boltzmann Constant	k	1.38E-23
Planck Constant	h	6.63E-34

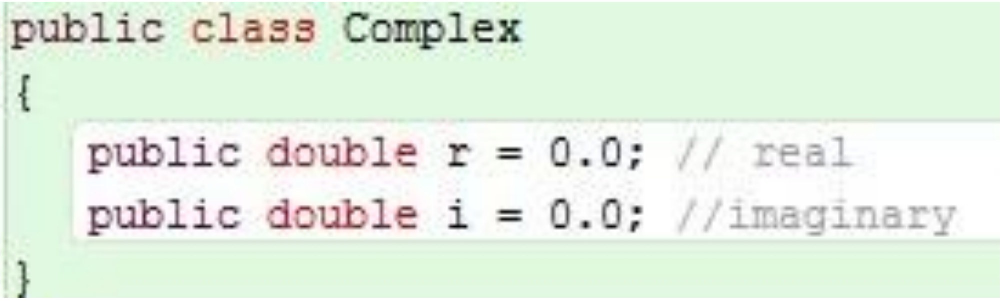
Use **PHY.C** for Speed of Light

Data Vector of Multiple-Tuple Class

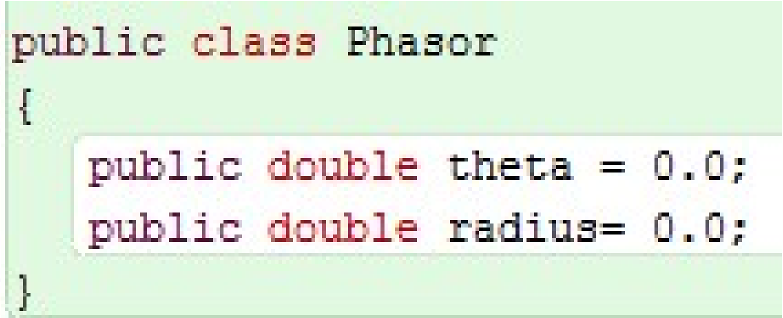
2D-Vector, Complex Number and Phasor



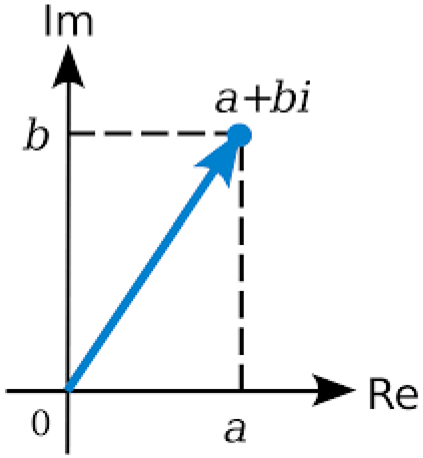
```
public class Vector
{
    public double x = 0.0;
    public double y = 0.0;
}
```



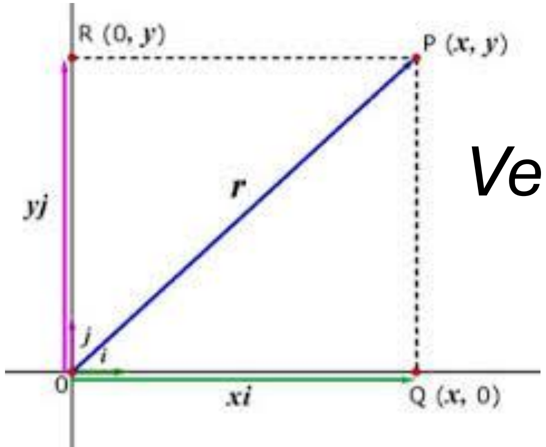
```
public class Complex
{
    public double r = 0.0; // real
    public double i = 0.0; //imaginary
}
```



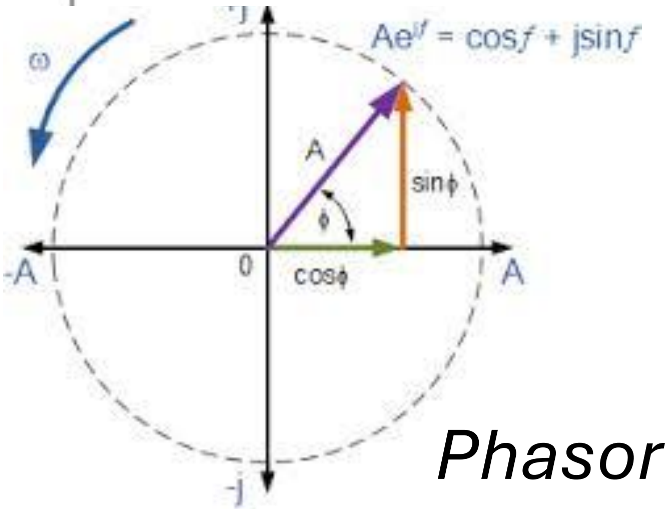
```
public class Phasor
{
    public double theta = 0.0;
    public double radius= 0.0;
}
```



Complex



Vector



Phasor

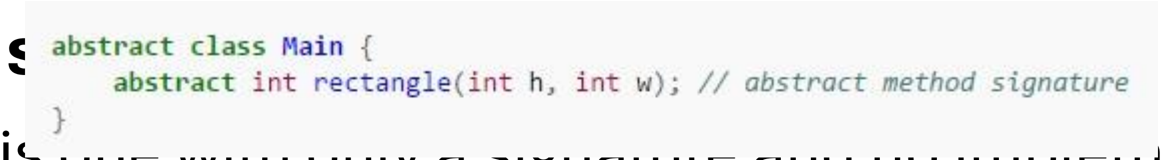
Class as Collection of Heterogeneous Data (Student)

```
class Student{
  String name        = “”;
  String studentID = “”; // or SSN
  String address     = “”;
  int[] score           = new int[6];
  char[] grade        = new char[6];
  ArrayList<String> classNames = new ArrayList<String>();
  double gpa = 0.0;
}
```

Methods

ways to access properties of a class

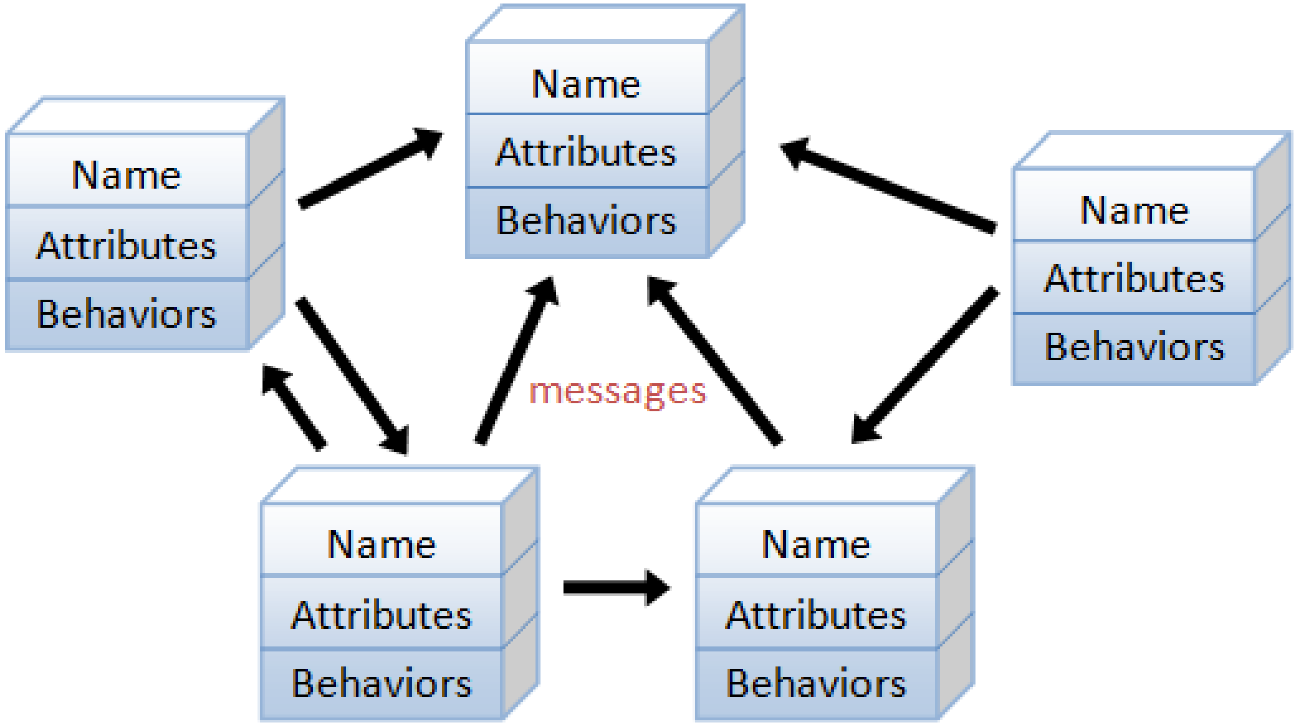
- **Functionality of Methods:**
- Getter method, Setter Method, Accessor, Mutator, Manager, Constructor, Destructor, Converter, Equality Checker, Identity Checker
- **Abstract Methods**



```
abstract class Main {
    abstract int rectangle(int h, int w); // abstract method signature
}
```

- An abstract method is one with only a signature and no implementation body. It is often used to specify that a subclass must provide an implementation of the method. Abstract methods are used to specify interfaces in some computer languages.
- **Class Methods/Instance Methods:** (a separate lecture)
- **Overloading/Overwriting** (a separate lecture)

OOP Environment



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Object-oriented programming (OOP) languages are designed to overcome these problems

- The basic unit of **OOP** is a **class**, which encapsulates both the static properties and dynamic operations within a "box", and specifies the public interface for using these boxes. Since classes are well-encapsulated, it is easier to reuse these classes. In other words, **OOP** combines **the data structures** and **algorithms** of a software entity inside the same box.
- OOP languages permit higher level of **abstraction** for solving real-life problems. The traditional procedural language (such as C and Pascal) forces you to think in terms of the structure of the computer (e.g. memory bits and bytes, array, decision, loop) rather than thinking in terms of the problem you are trying to solve. The OOP languages (such as Java, C++ and C#) let you think in the problem space, and use software objects to represent and abstract entities of the problem space to solve the problem.

Demonstration Program

PHY.JAVA+VECTOR.JAVA+PHASOR.JAVA+

COMPLEX.JAVA+COMPLEXTEST.JAVA

Reference Variables

Lecture 5

Reference Data Fields

- The data fields can be of reference types. For example, the following **Student** class contains a data field **name** of the **String** type.

```
public class Student {
  String name; // name has default value null
  int age; // age has default value 0
  boolean isScienceMajor; // isScienceMajor has default value
false
  char gender; // c has default value '\u0000'
}
```

The null Value

- If a data field of a reference type does not reference any object, the data field holds a special literal value, **null**.

- The default value of a data field is **null** for a reference type, 0 for a numeric type, false for a boolean type, and '\u0000' for a char type. However, Java assigns no default value to a local variable inside a method.

```
public class Test {
  public static void main(String[] args) {
    Student student = new Student();
    System.out.println("name? " + student.name);
    System.out.println("age? " + student.age);
    System.out.println("isScienceMajor? " +
                        student.isScienceMajor);
    System.out.println("gender? " + student.gender);
  }
}
```

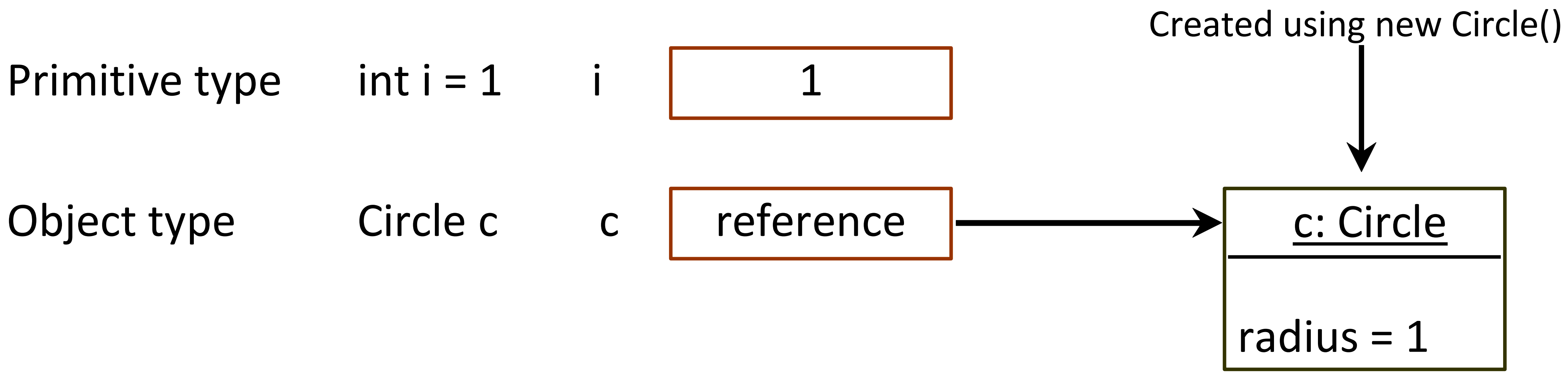
Example

- Java assigns no default value to a local variable inside a method.

```
public class Test {
  public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
  }
}
```

Compilation error: variables not initialized

Differences between Variables of Primitive Data Types and Object Types



Created using new Circle()

Primitive type int i = 1 i 1

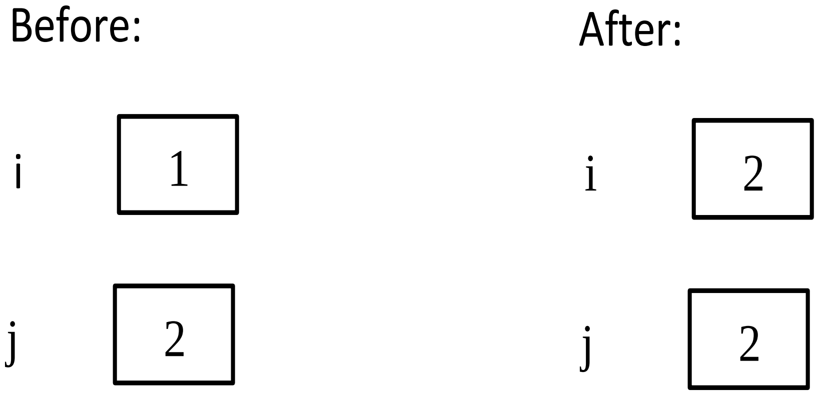
Object type Circle c c reference

c: Circle

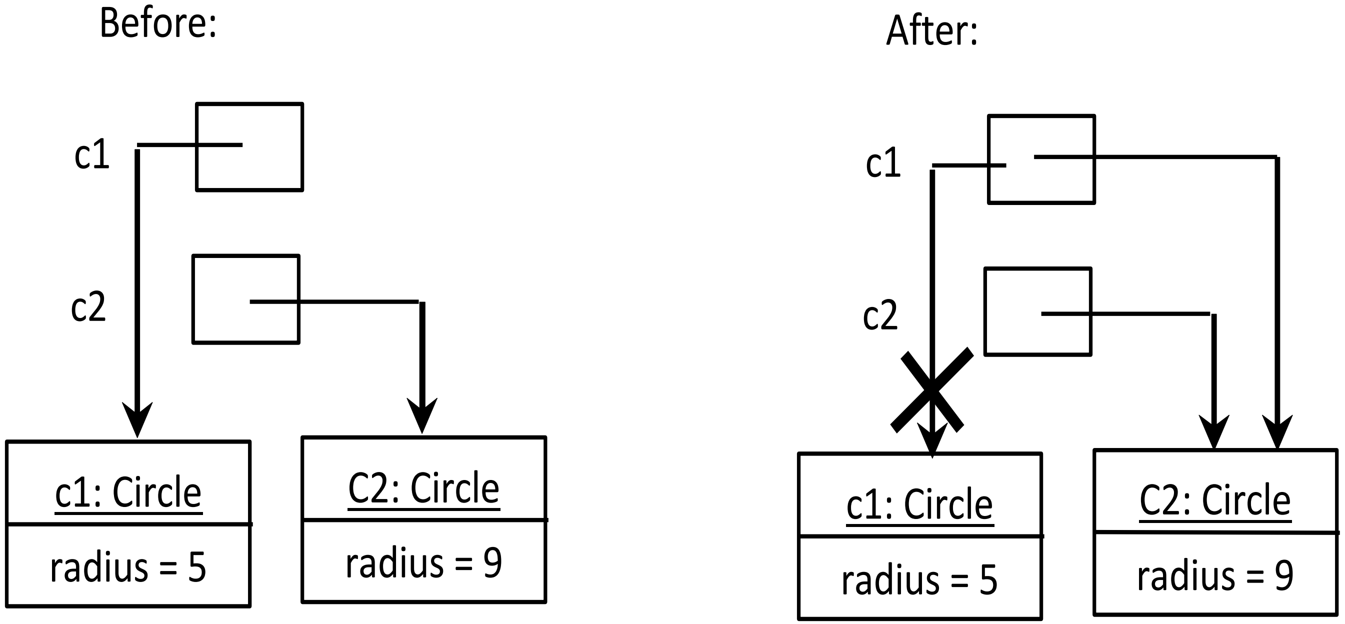
radius = 1

Copying Variables of Primitive Data Types and Object Types

Primitive type assignment i = j



Object type assignment c1 = c2



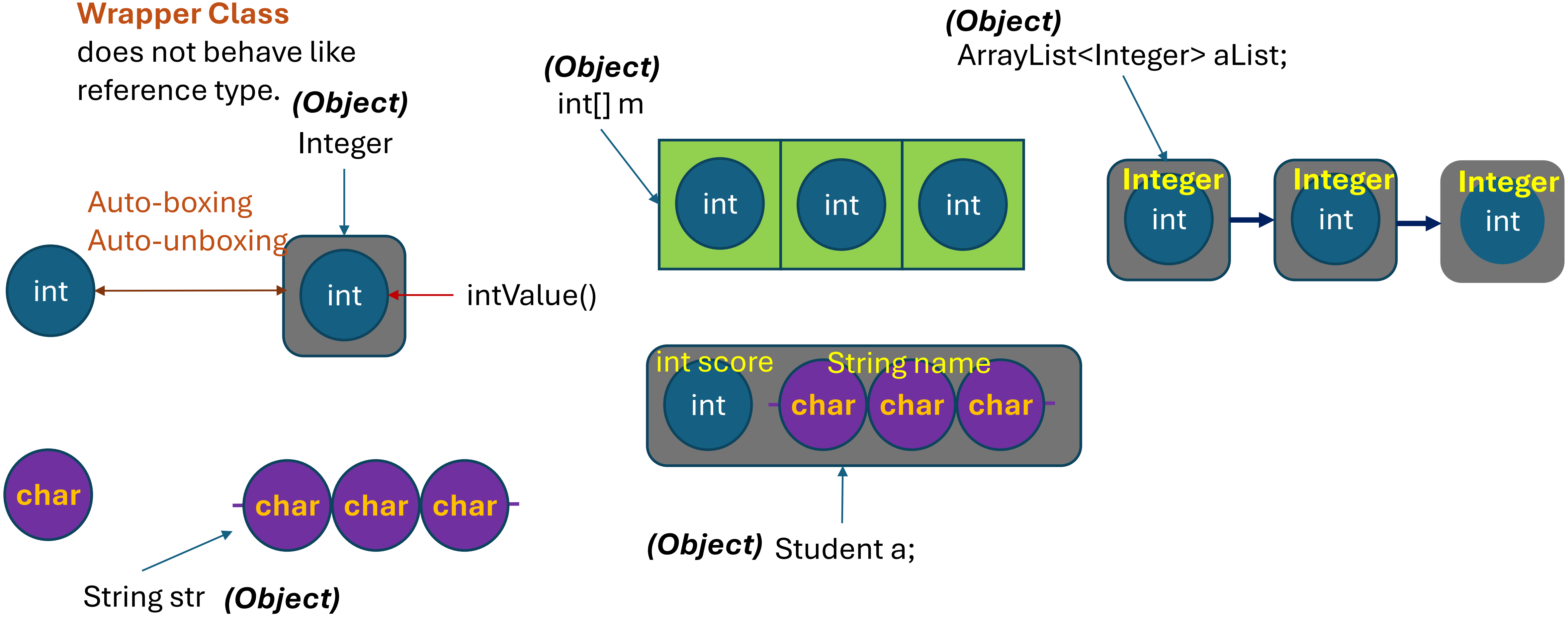
Garbage Collection

- As shown in the previous figure, after the assignment statement c1 = c2, c1 points to the same object referenced by c2. The object previously referenced by c1 is no longer referenced. This object is known as **garbage** (dangling object).
- Garbage is automatically collected by **JVM**.

Garbage Collection, cont.

TIP: If you know that an object is no longer needed, you can explicitly assign **null** to a reference variable for the object. The **JVM** will automatically collect the space if the object is not referenced by any variable .

Primitive Data Type V.S. Reference Data Type



Wrapper Class

does not behave like reference type.

(Object)

Integer

Auto-boxing

Auto-unboxing

intValue()

(Object)

int[] m

(Object)

ArrayList<Integer> aList;

(Object) Student a;

String str ***(Object)***

Objectives

- Class Classification: Data, Utility, Wrapper, Functional, Tester, and Handler
- Data Classes
- Classes Tested 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 6

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 getName(String n){ name = n; }
   public void getStudentID(int id){ studentID= id;  }
   public void getMathScore(int s) { mathScore=s; }
   public void getEngishScore(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 7

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

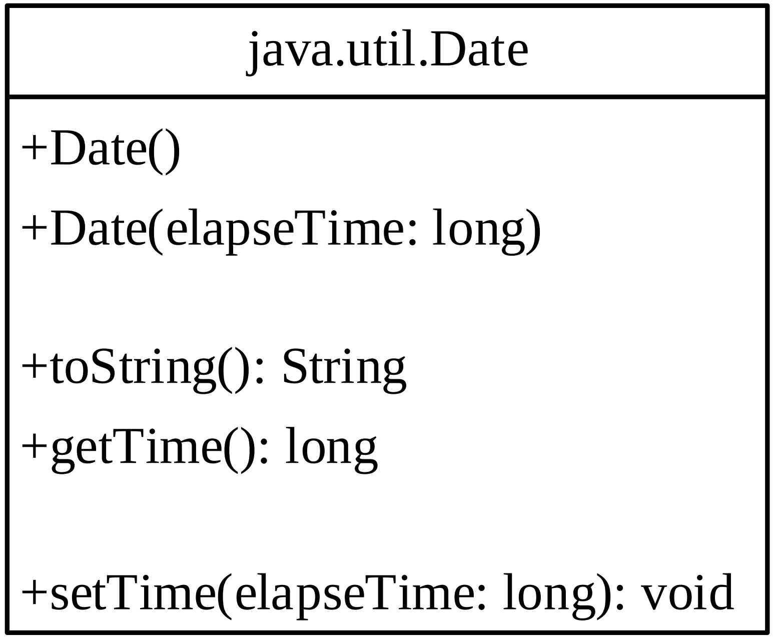
Lecture 8

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 modifer



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

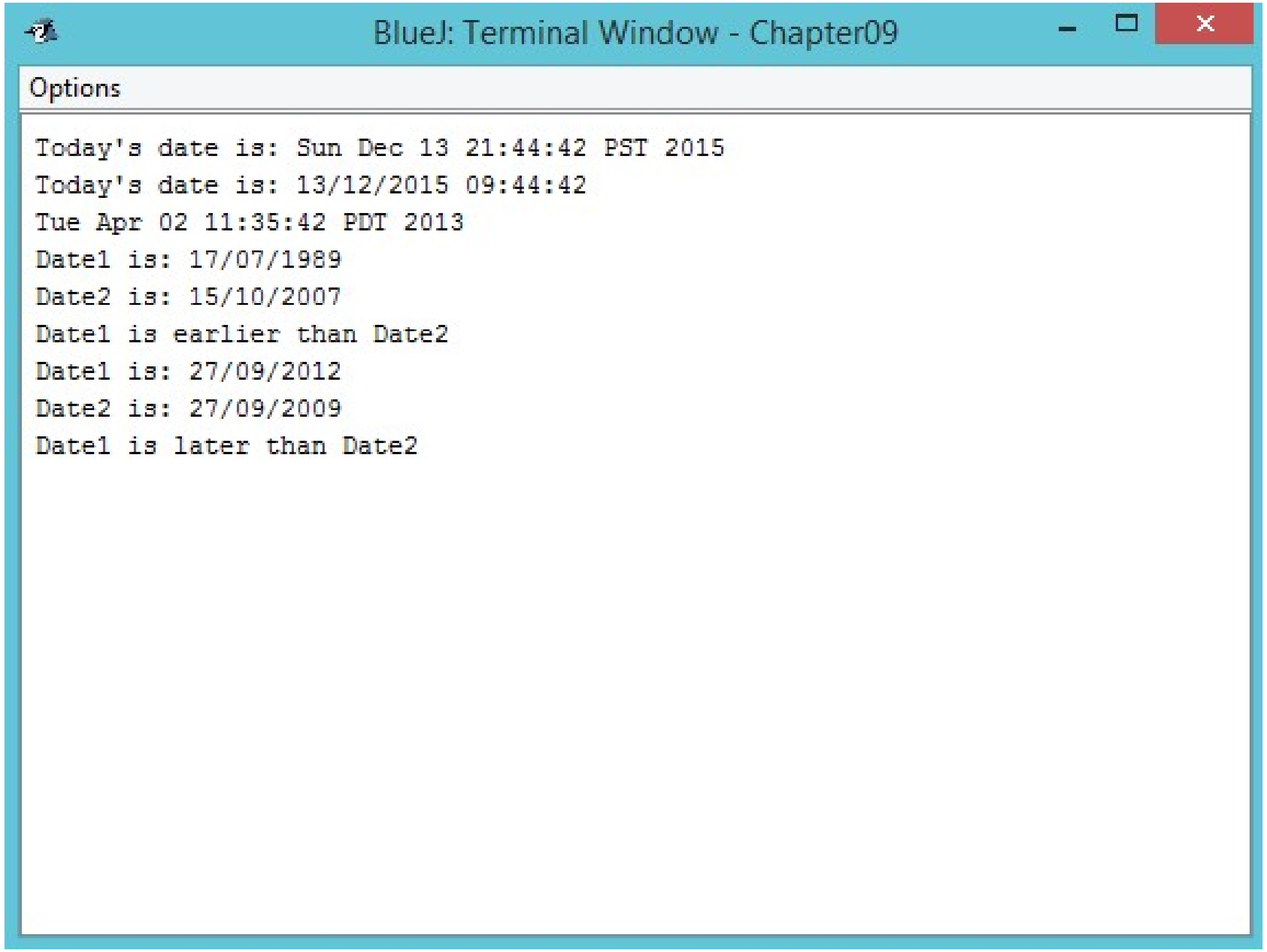
```
java.util.Date date = new
   java.util.Date();
System.out.println(date.toString());
```

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

Go BlueJ.

Demonstration Program

DATEEXAMPLE.JAVA



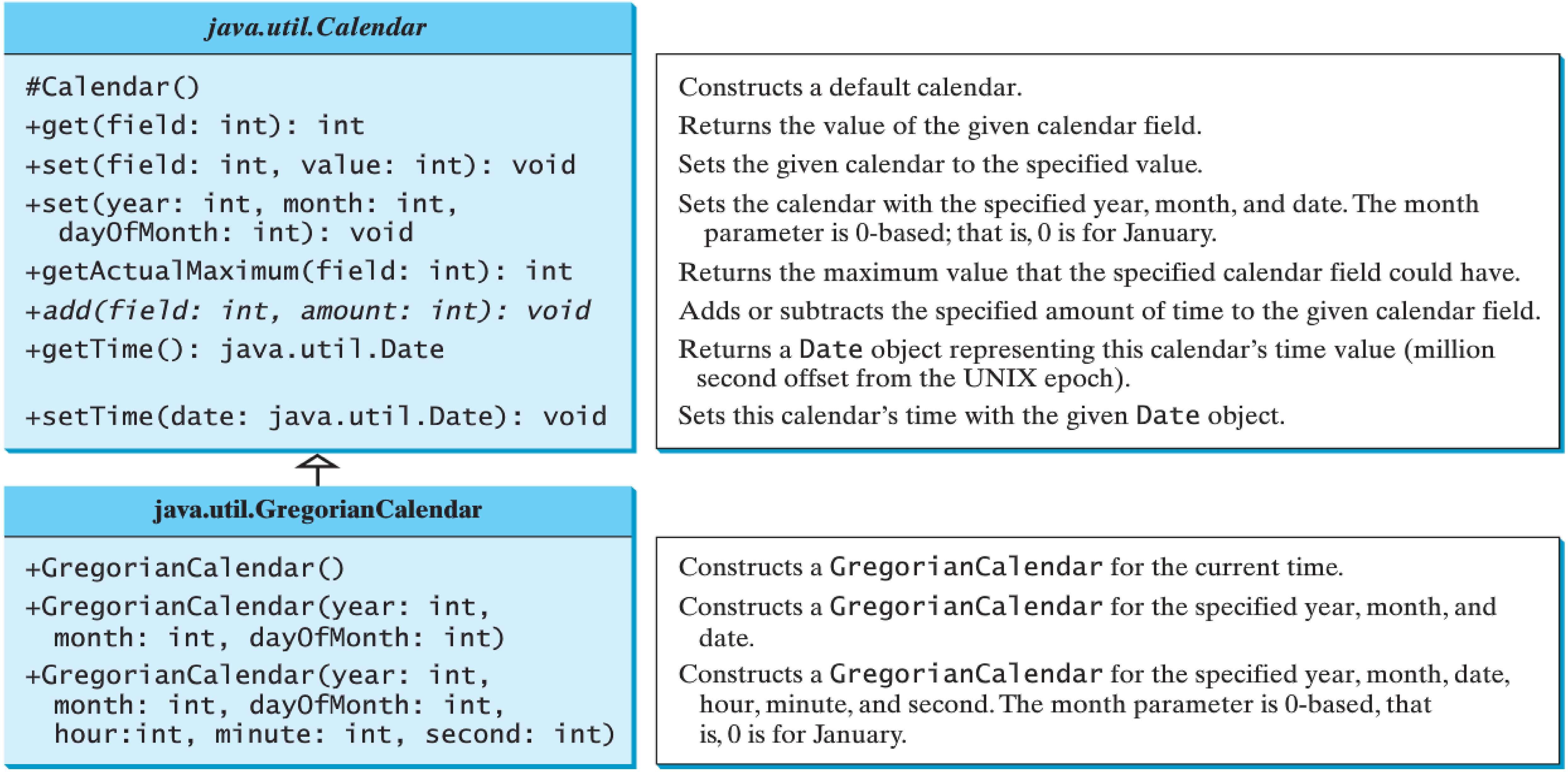
```
Today's date is: Sun Dec 13 21:44:42 PST 2015
Today's date is: 13/12/2015 09:44:42
Tue Apr 02 11:35:42 PDT 2013
Date1 is: 17/07/1989
Date2 is: 15/10/2007
Date1 is earlier than Date2
Date1 is: 27/09/2012
Date2 is: 27/09/2009
Date1 is later than Date2
```

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)

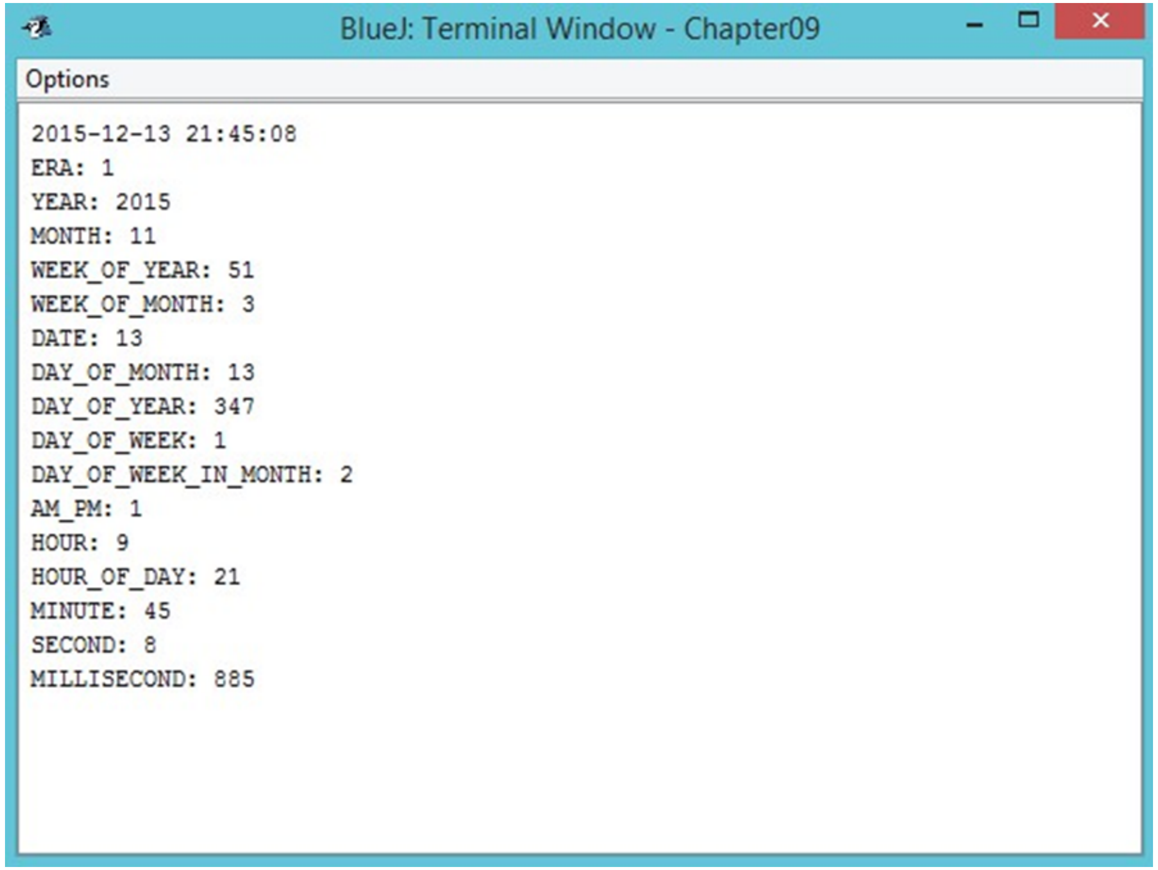
Constructs a `GregorianCalendar` for the current time.

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



```
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

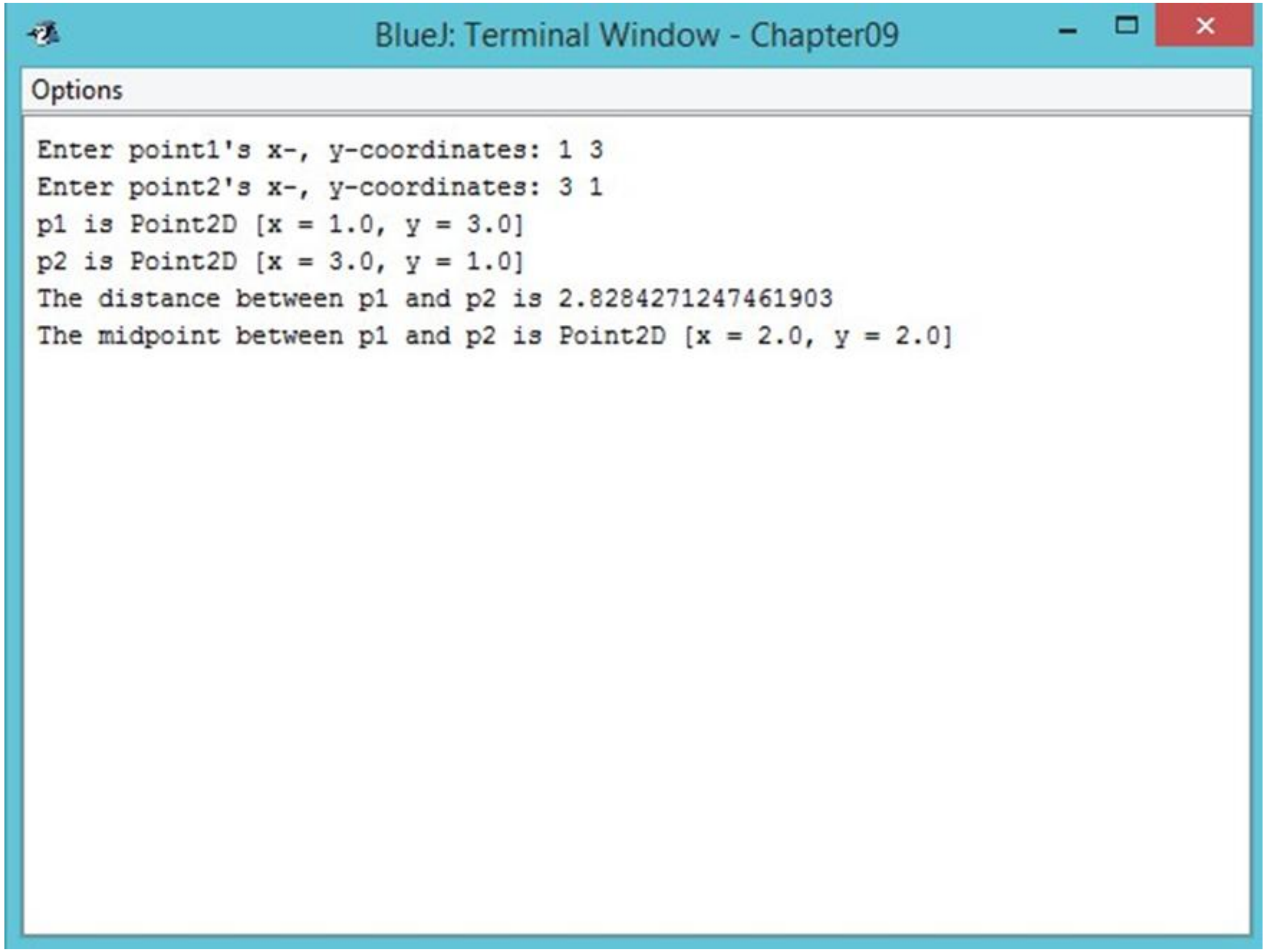
Point2D Class

- 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



```
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 9

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:

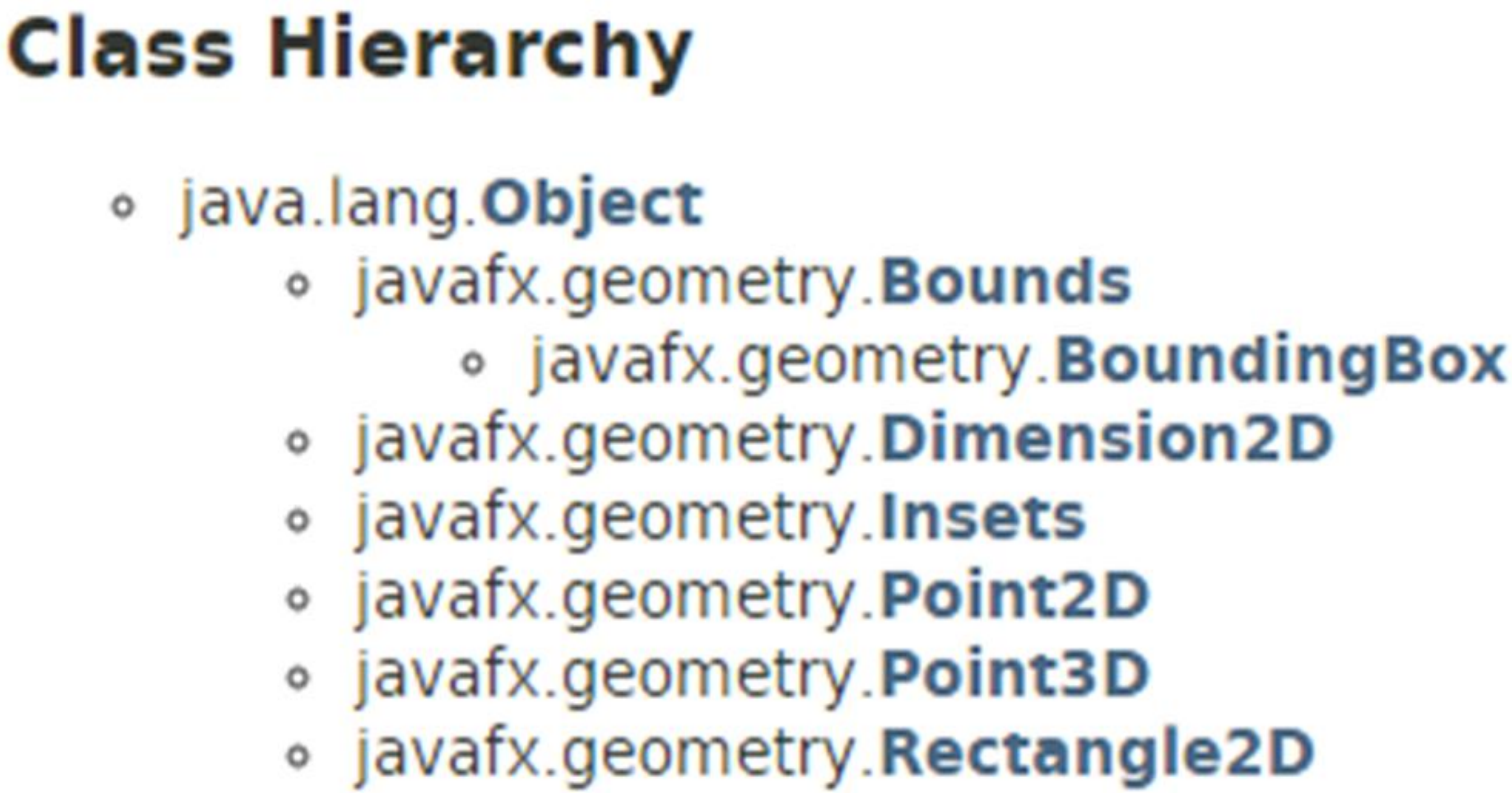
javafx package (GUI)
java.awt package (GUI)
java.swing package (GUI)

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:** Note 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 10

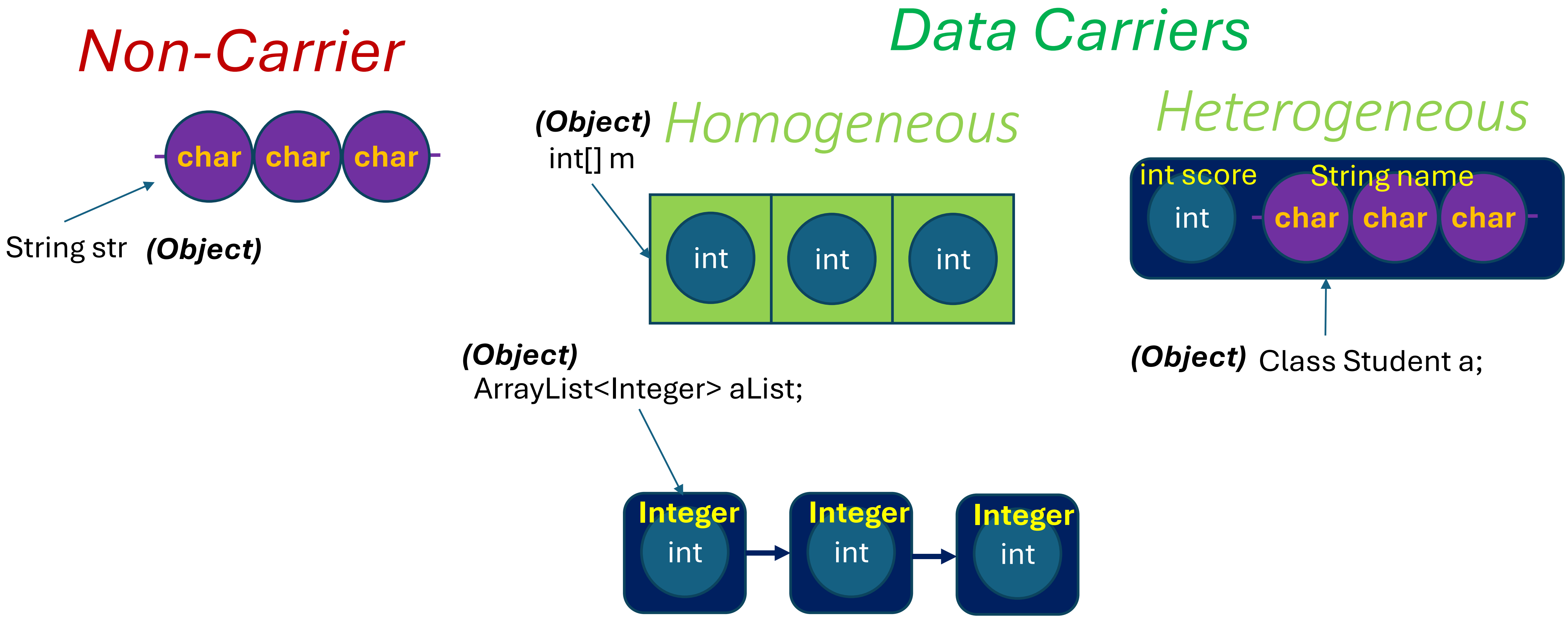
Object is a Heterogeneous Data Record

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

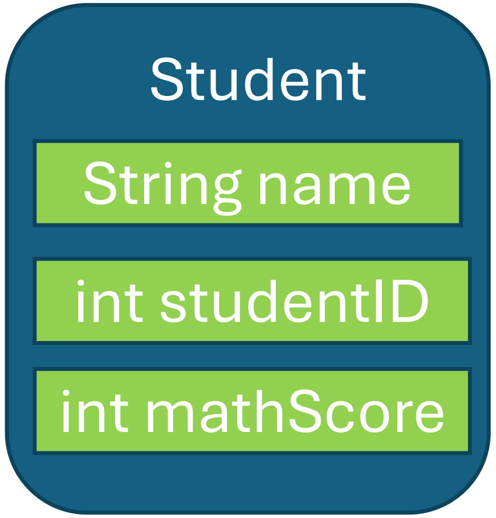
Homogeneous

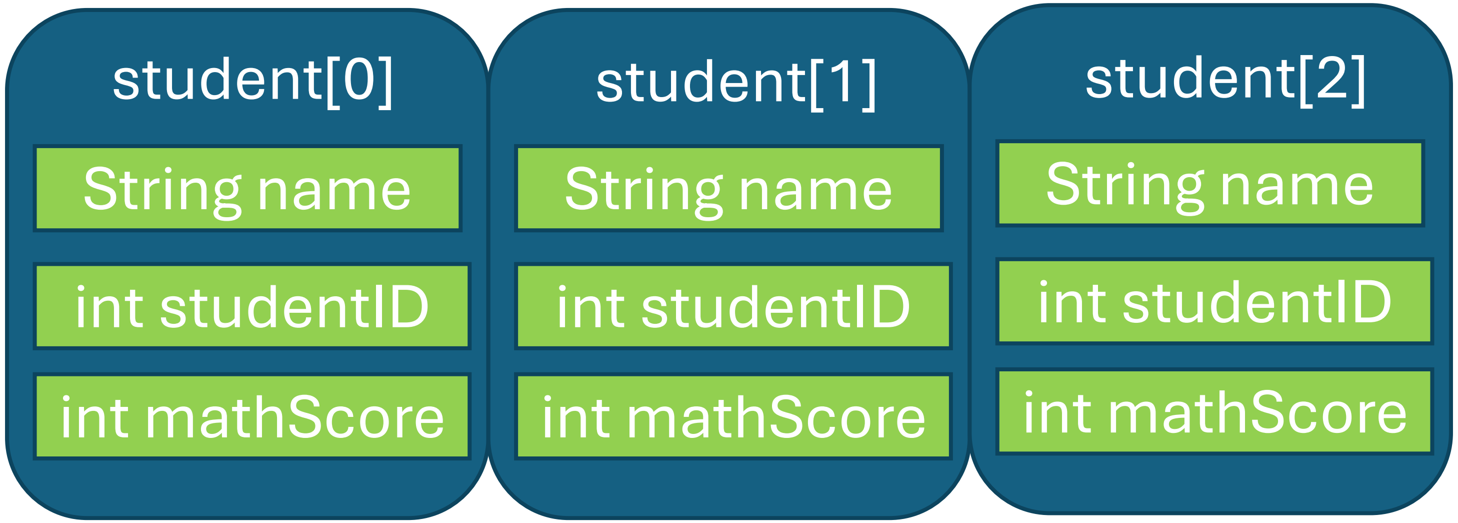
Heterogeneous

Array of Objects

Student Class

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

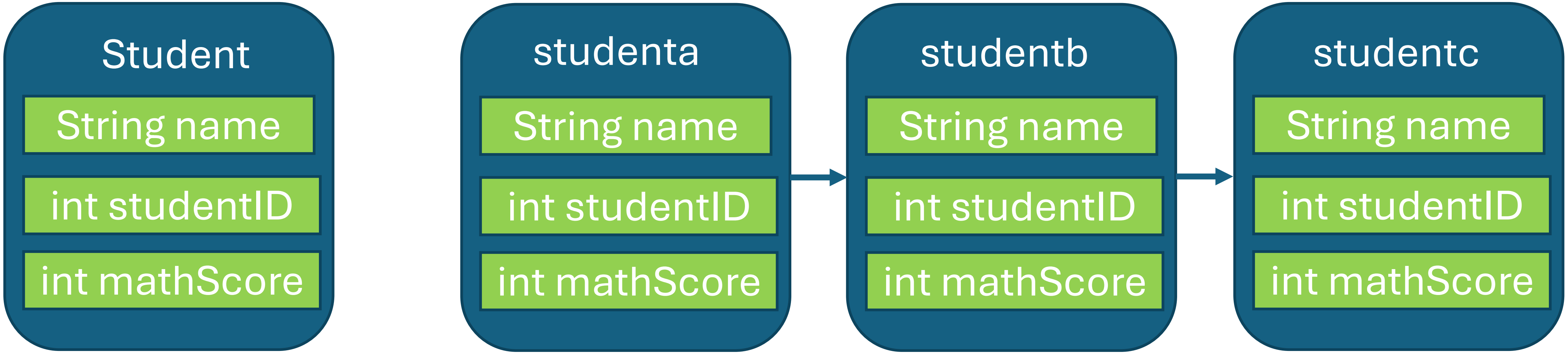




ArrayList of Objects

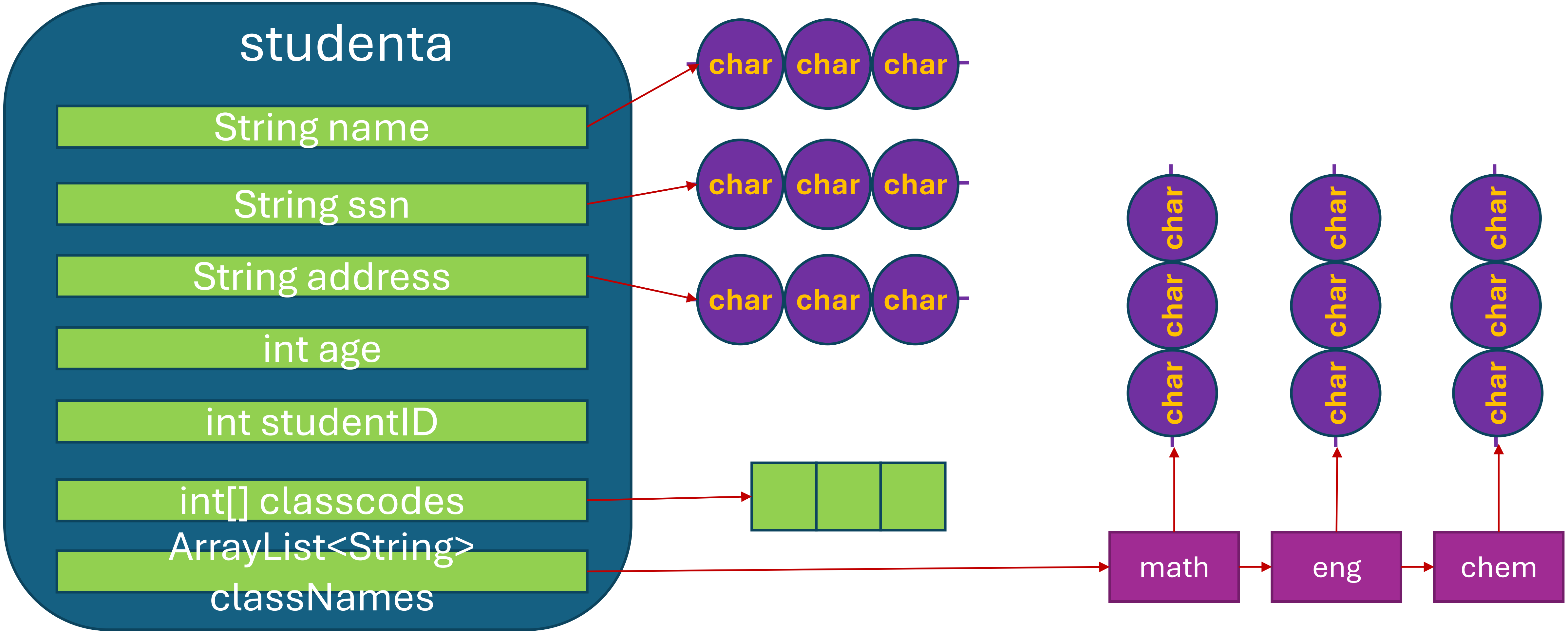
Student Class ArrayList<Student> al = new ArrayList<Student>

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 11

Array of Objects (ArrayList of Objects)

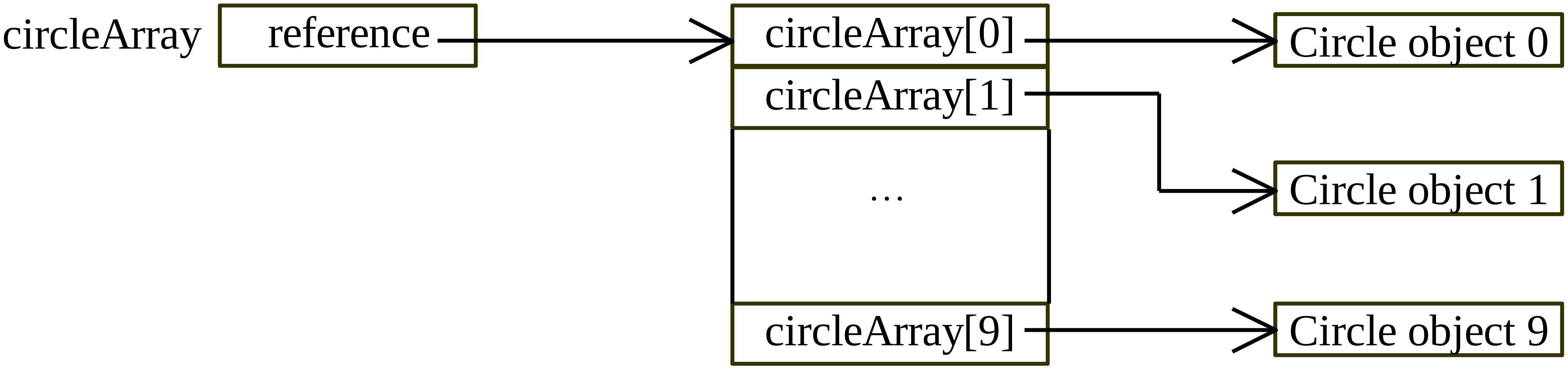
```
Circle[] circleArray = new Circle[10];
ArrayList<Circle> circleArrayList = new ArrayList<Circle>();
```

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

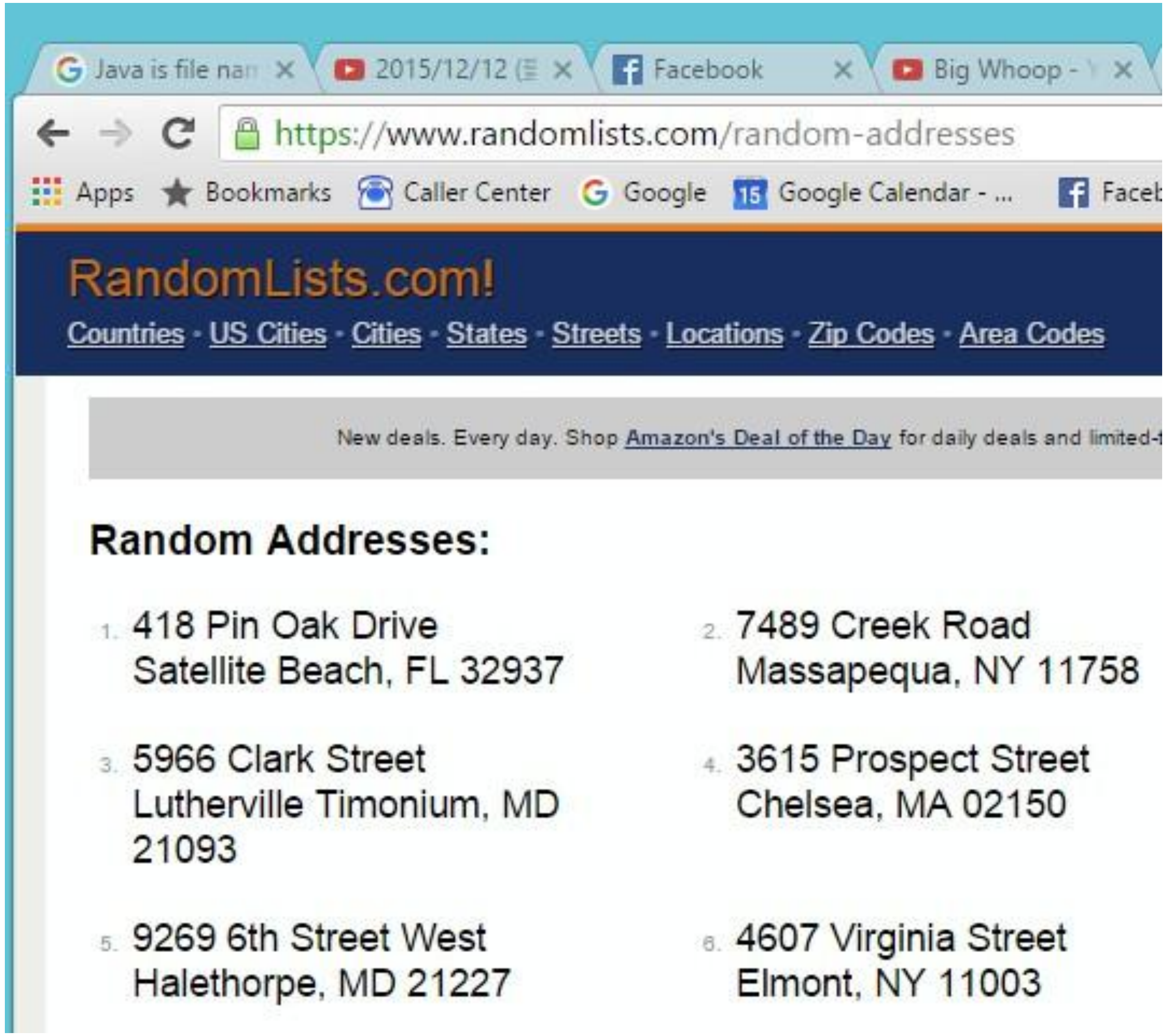
(2) Newly added features:

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.)

Public Domain Random Data Generators

Random Address Generator



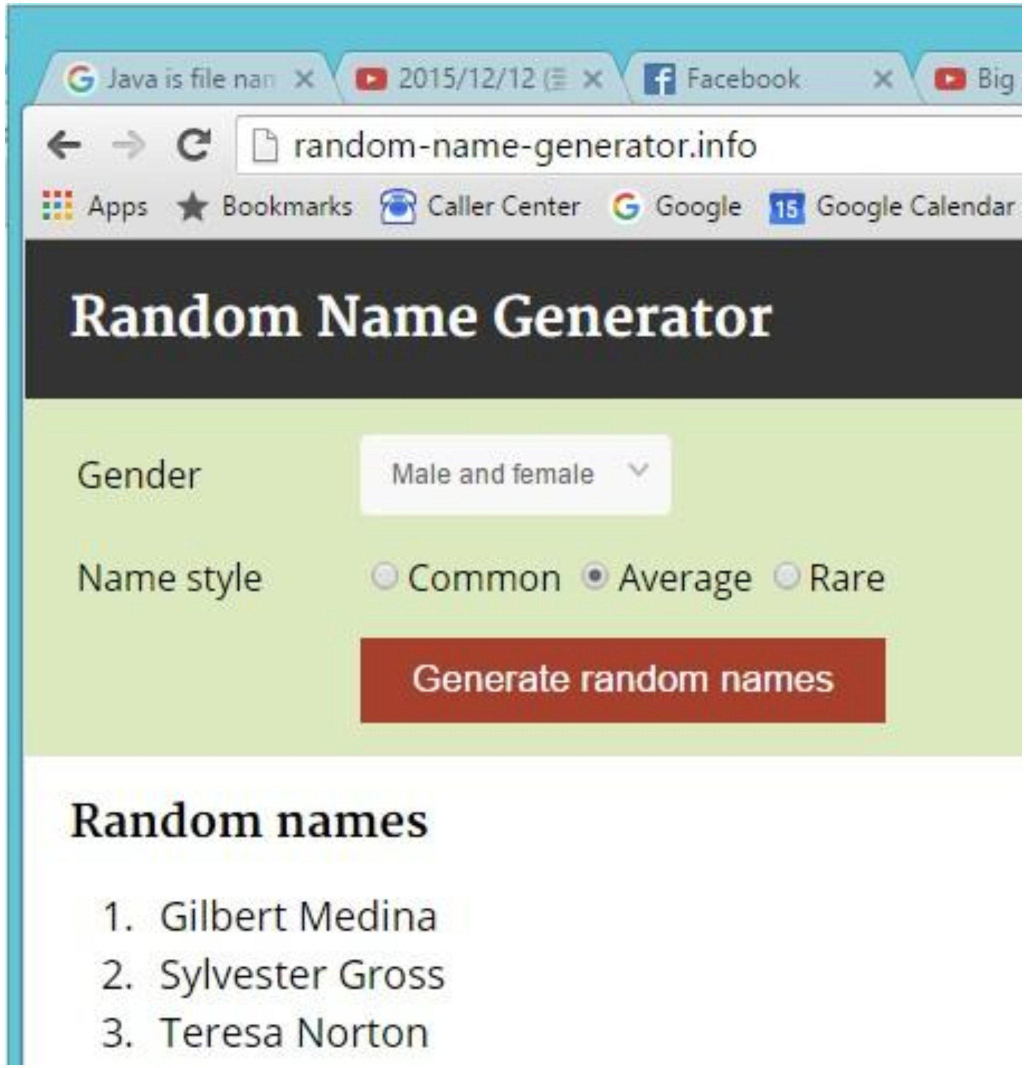
RandomLists.com!

Countries · US Cities · Cities · States · Streets · Locations · Zip Codes · Area Codes

Random Addresses:

1. 418 Pin Oak Drive
   Satellite Beach, FL 32937
2. 7489 Creek Road
   Massapequa, NY 11758
3. 5966 Clark Street
   Lutherville Timonium, MD 21093
4. 3615 Prospect Street
   Chelsea, MA 02150
5. 9269 6th Street West
   Halethorpe, MD 21227
6. 4607 Virginia Street
   Elmont, NY 11003

Public Domain Random Data Generators **Random Name Generator**



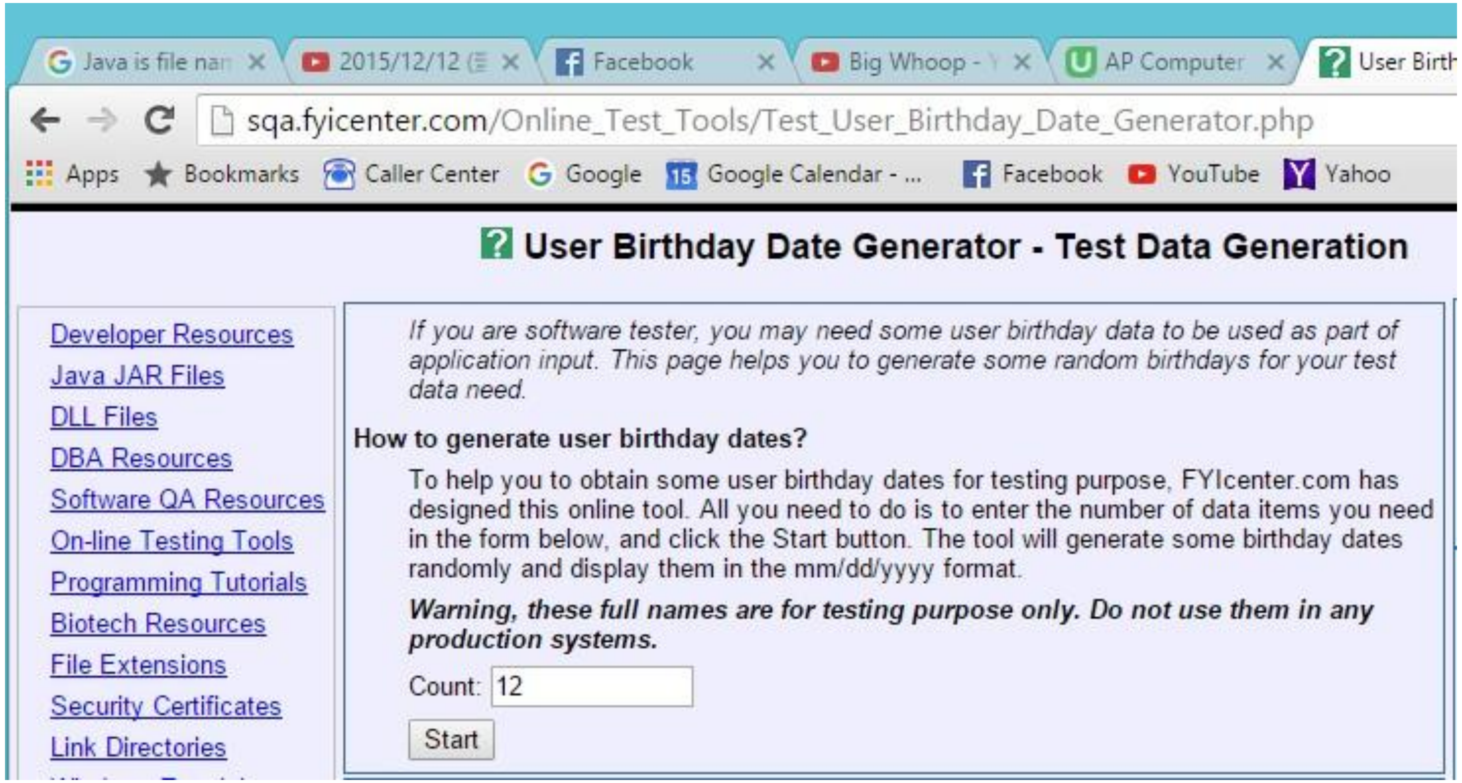
Random Name Generator

Gender: Male and female

Name style: ○ Common ● Average ○ Rare

Random names

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



User Birthday Date Generator - Test Data Generation

If you are software tester, you may need some user birthday data to be used as part of application input. This page helps you to generate some random birthdays for your test data need.

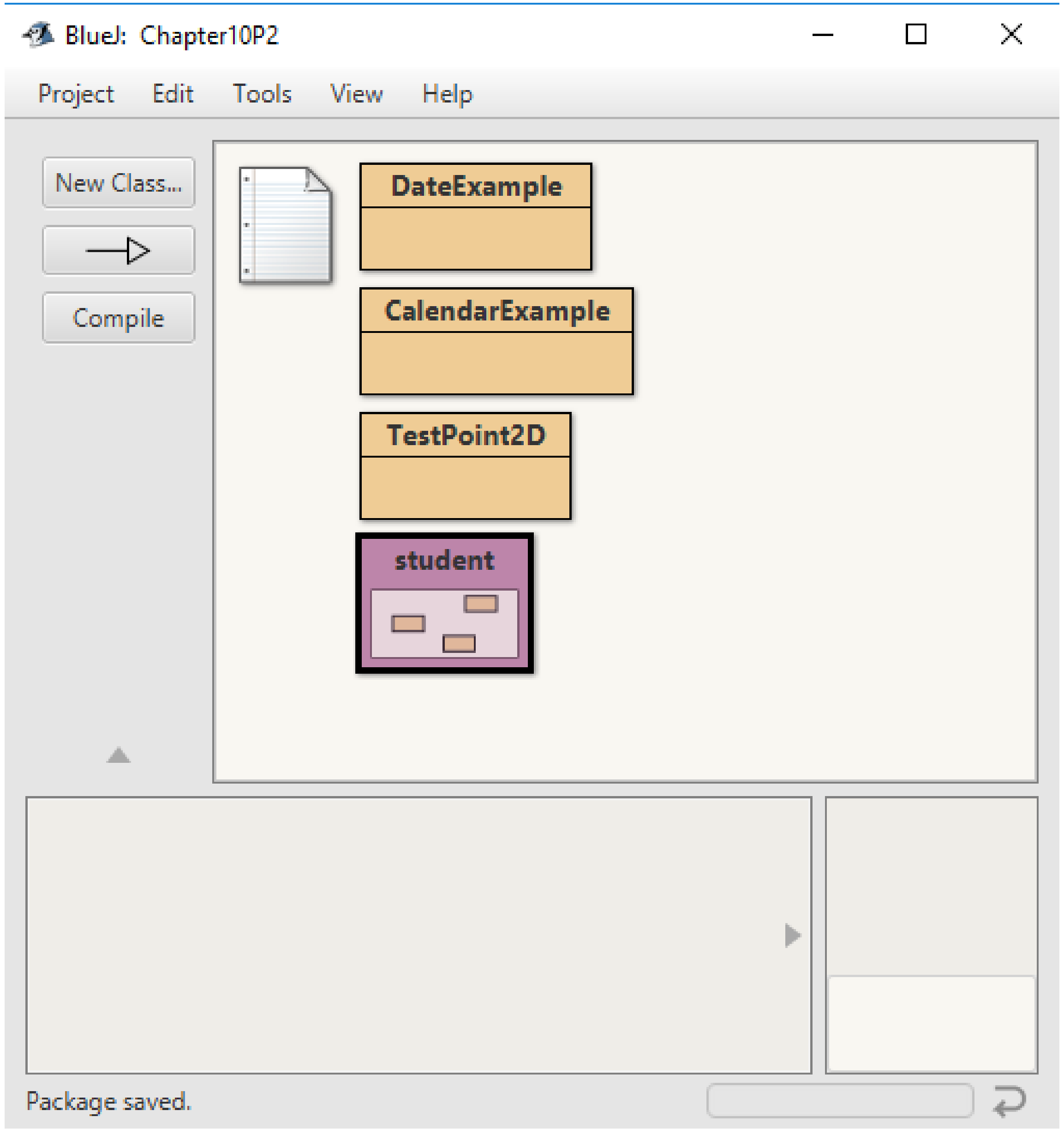
How to generate user birthday dates?

To help you to obtain some user birthday dates for testing purpose, FYIcenter.com has designed this online tool. All you need to do is to enter the number of data items you need in the form below, and click the Start button. The tool will generate some birthday dates randomly and display them in the mm/dd/yyyy format.

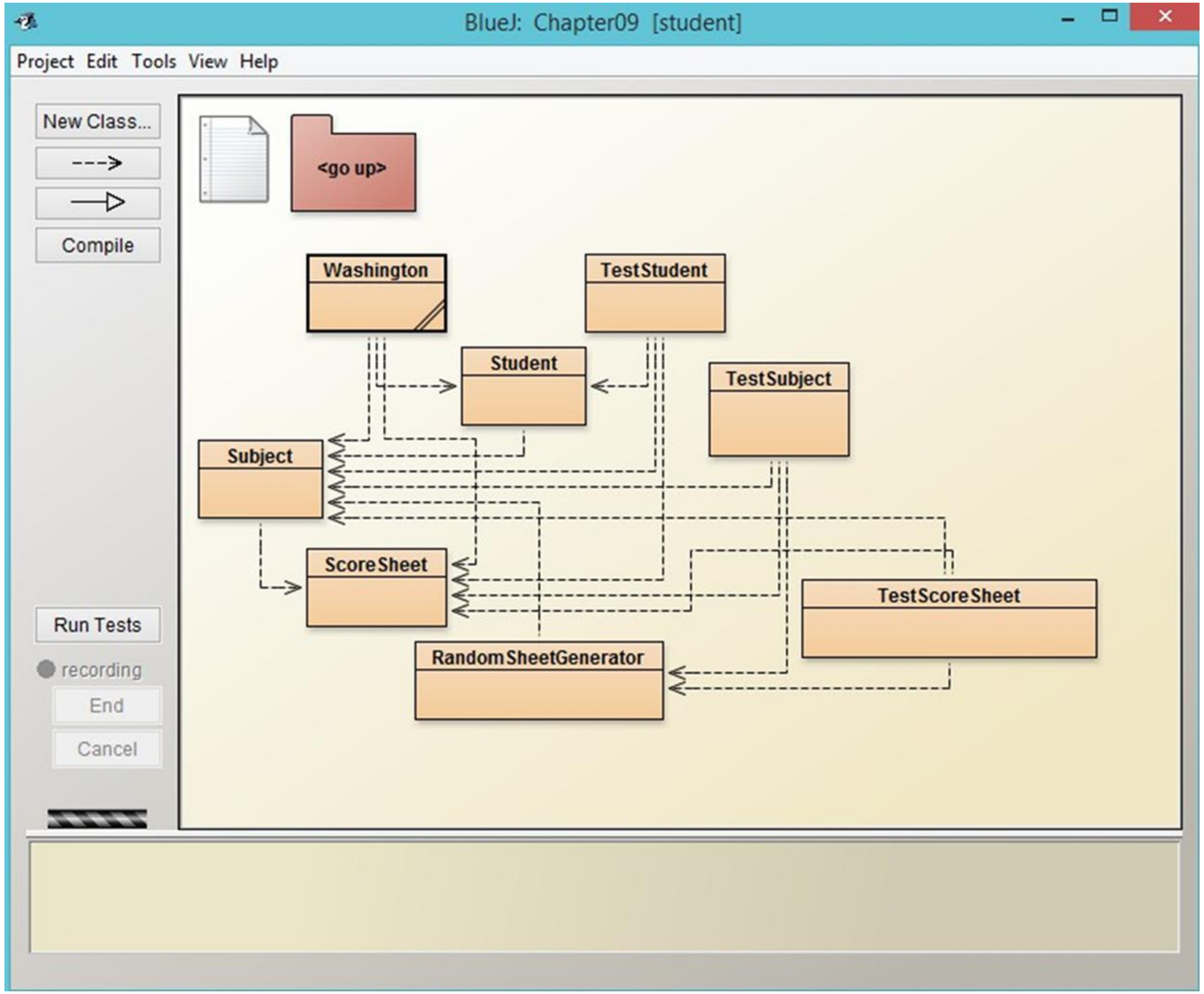
Warning, these full names are for testing purpose only. Do not use them in any production systems.

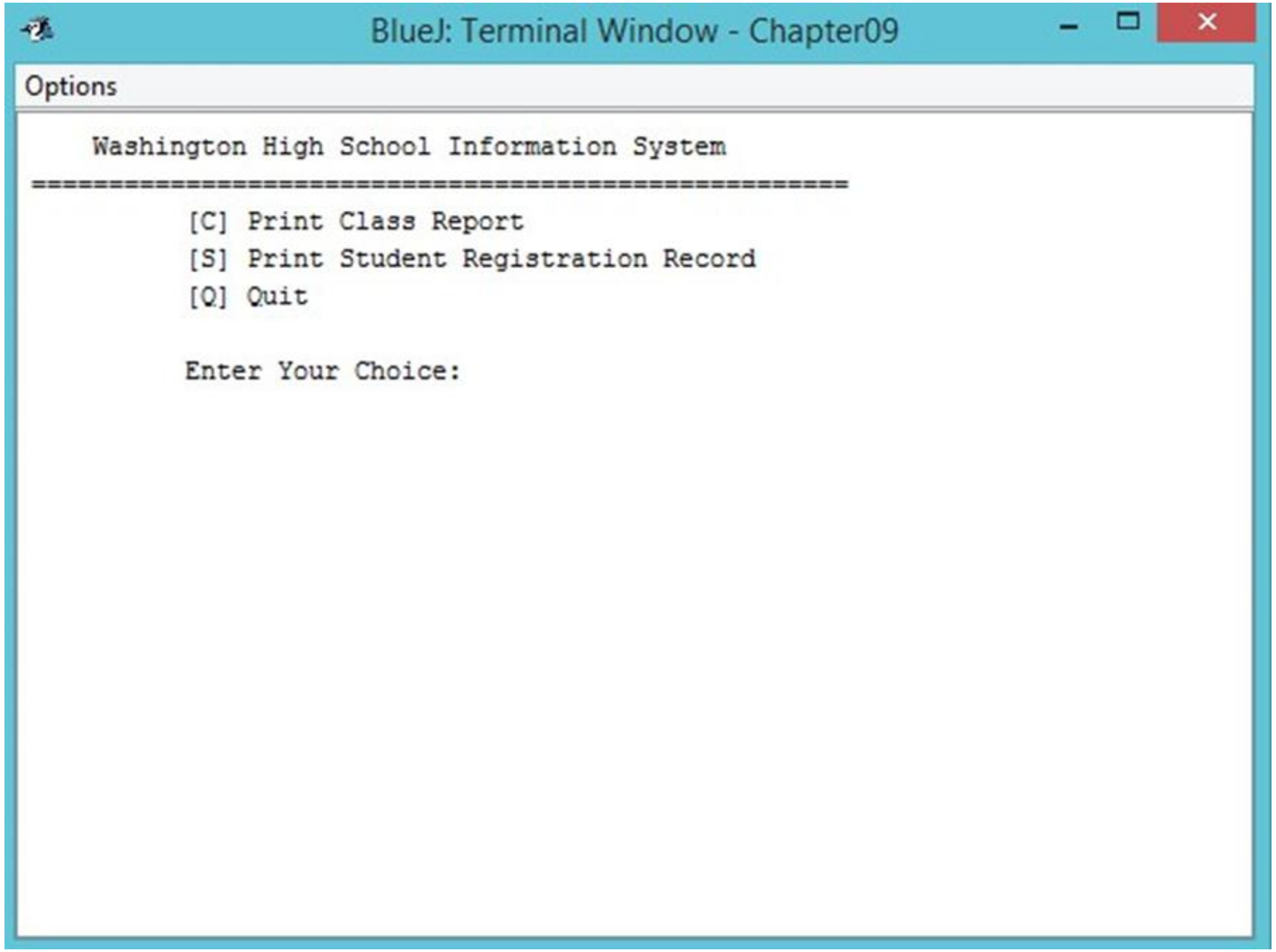
Count: 12

Public Domain Random Data Generators **Random Birthday Generator**



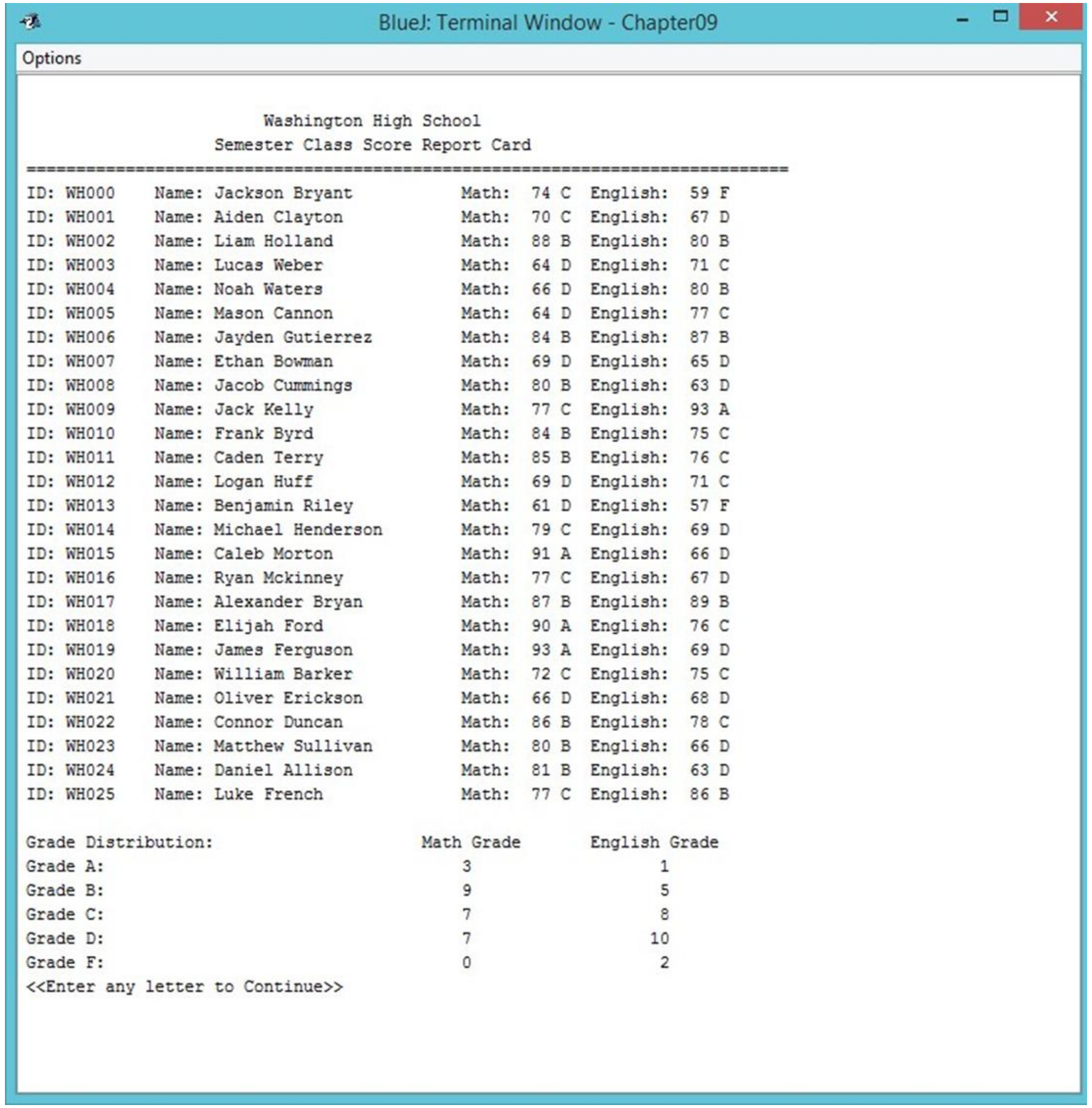
Package saved.


BlueJ: Chapter09 [student]
Project
Edit
Tools
View
Help
New Class...
Compile
Run Tests
recording
End
Cancel
<go up>
Washington
TestStudent
Student
TestSubject
Subject
ScoreSheet
TestScoreSheet
RandomSheetGenerator

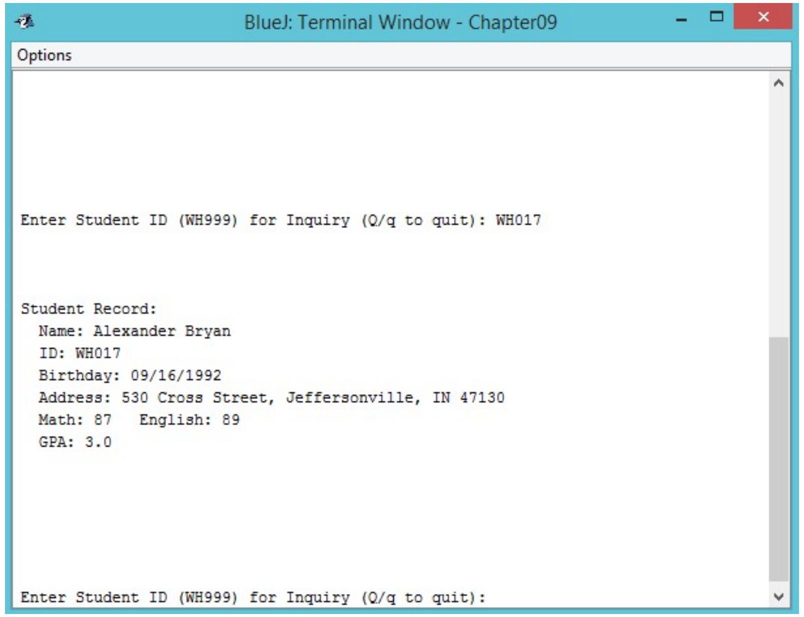


```
    Washington High School Information System
=====================================================
          [C] Print Class Report
          [S] Print Student Registration Record
          [Q] Quit

          Enter Your Choice:
```

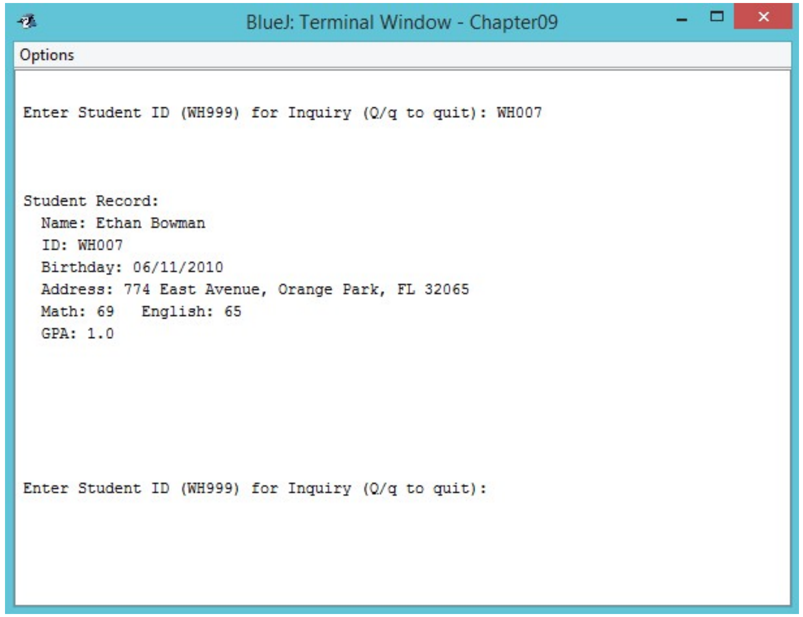
Washington High School **Welcome Manual**



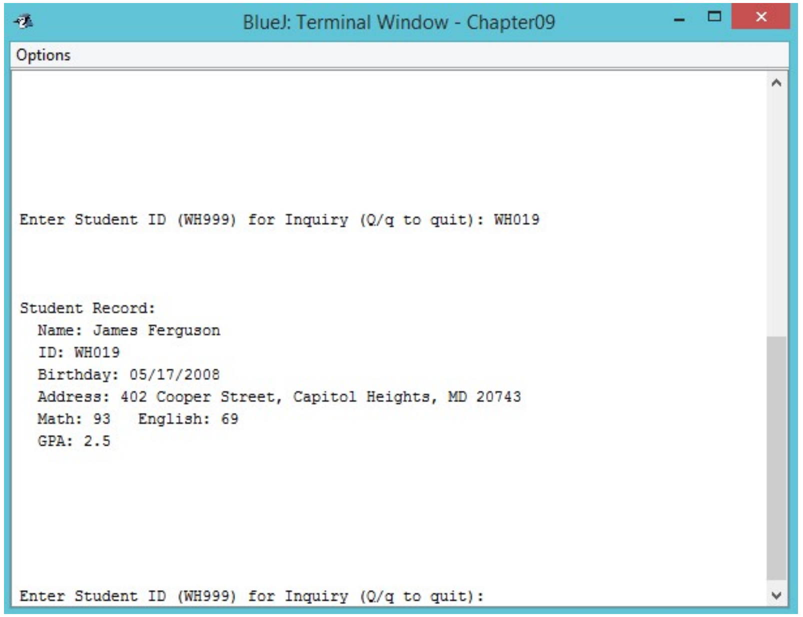
```
                        Washington High School
                   Semester Class Score Report Card
=============================================================================
ID: WH000    Name: Jackson Bryant           Math:  74 C  English:  59 F
ID: WH001    Name: Aiden Clayton            Math:  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>>
```



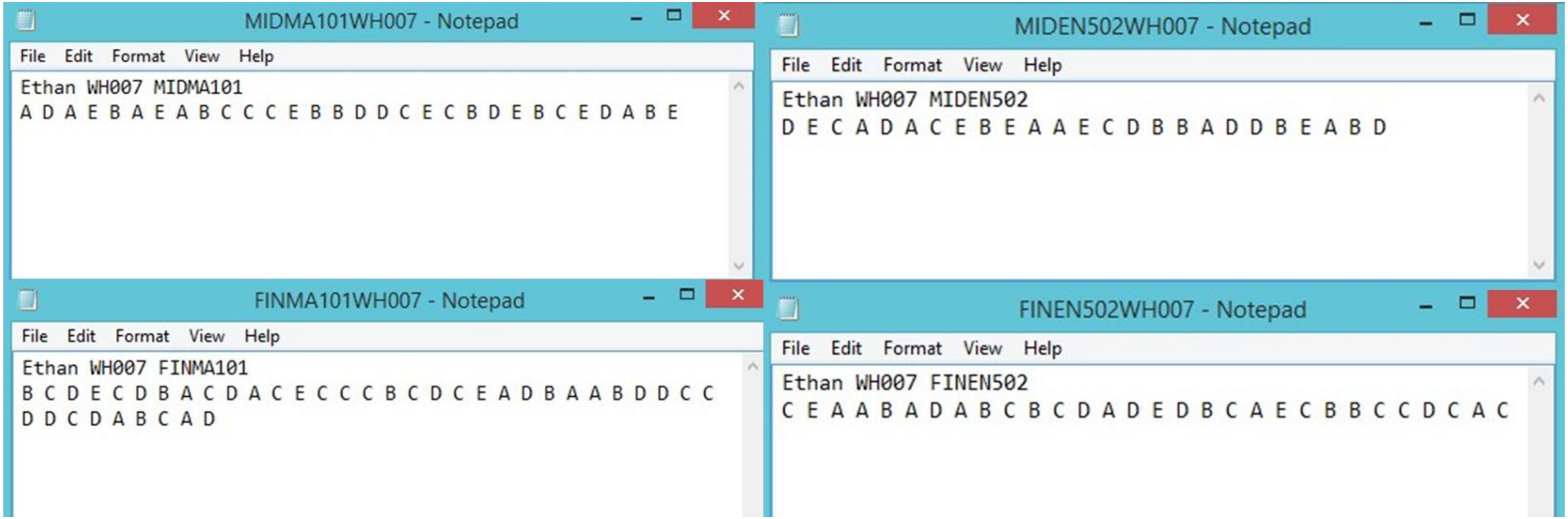
```
Enter Student ID (WH999) for Inquiry (Q/q to quit): WH017



Student Record:
  Name: Alexander Bryan
  ID: WH017
  Birthday: 09/16/1992
  Address: 530 Cross Street, Jeffersonville, IN 47130
  Math: 87   English: 89
  GPA: 3.0




Enter Student ID (WH999) for Inquiry (Q/q to quit):
```



```
Enter Student ID (WH999) for Inquiry (Q/q to quit): WH007



Student Record:
  Name: Ethan Bowman
  ID: WH007
  Birthday: 06/11/2010
  Address: 774 East Avenue, Orange Park, FL 32065
  Math: 69   English: 65
  GPA: 1.0




Enter Student ID (WH999) for Inquiry (Q/q to quit):
```



```
Enter Student ID (WH999) for Inquiry (Q/q to quit): WH019



Student Record:
  Name: James Ferguson
  ID: WH019
  Birthday: 05/17/2008
  Address: 402 Cooper Street, Capitol Heights, MD 20743
  Math: 93   English: 69
  GPA: 2.5




Enter Student ID (WH999) for Inquiry (Q/q to quit):
```


MIDMA101WH007 - Notepad
File Edit Format View Help
MIDEN502WH007 - Notepad
File Edit Format View Help
FINMA101WH007 - Notepad
File Edit Format View Help
FINEN502WH007 - Notepad
File Edit Format View Help

```
Ethan WH007 MIDMA101
A D A E B A E A B C C C E B B D D C E C B D E B C E D A B E
```

```
Ethan WH007 MIDEN502
D E C A D A C E B E A A E C D B B A D D B E A B D
```

```
Ethan WH007 FINMA101
B C D E C D B A C D A C E C C C B C D C E A D B A A B D D C C
D D C D A B C A D
```

```
Ethan WH007 FINEN502
C E A A B A D A B C B C D A D E D B C A E C B B C C D C A C
```

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).

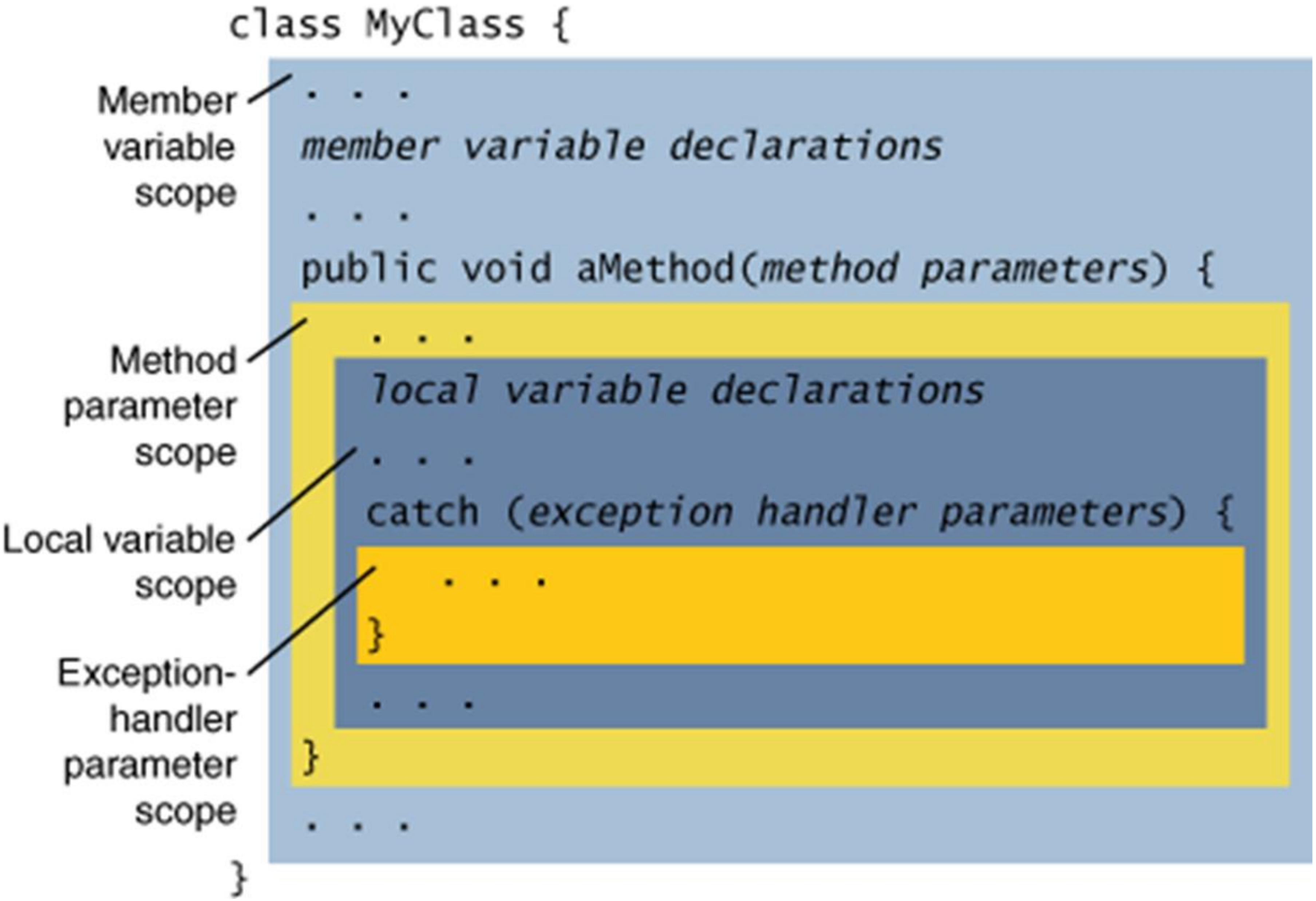
(3) Implement from Score Sheet and Random Score Sheet Generator first. Then, Subject Class, Student Class and finally the Washington Class.

Demo Program: Washington Project

- Student should work on this project in Class.
- Or, as a take-out lab project.

Scope of Members

Lecture 12



```
class MyClass {
    . . .
    member variable declarations
    . . .
    public void aMethod(method parameters) {
        . . .
        local variable declarations
        . . .
        catch (exception handler parameters) {
            . . .
        }
        . . .
    }
    . . .
}
```

Member variable scope

Method parameter scope

Local variable scope

Exception-handler parameter scope

Local Variable Versus Global Variable

Global Variables:

Member Properties:

Instance Variable - double radius;

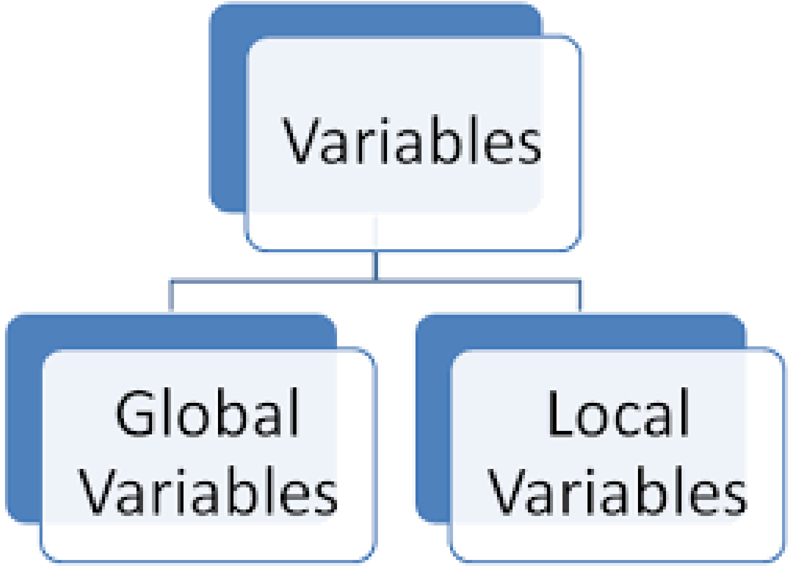
Class Variable (static) – static int num;

Local Variables:

Arguments

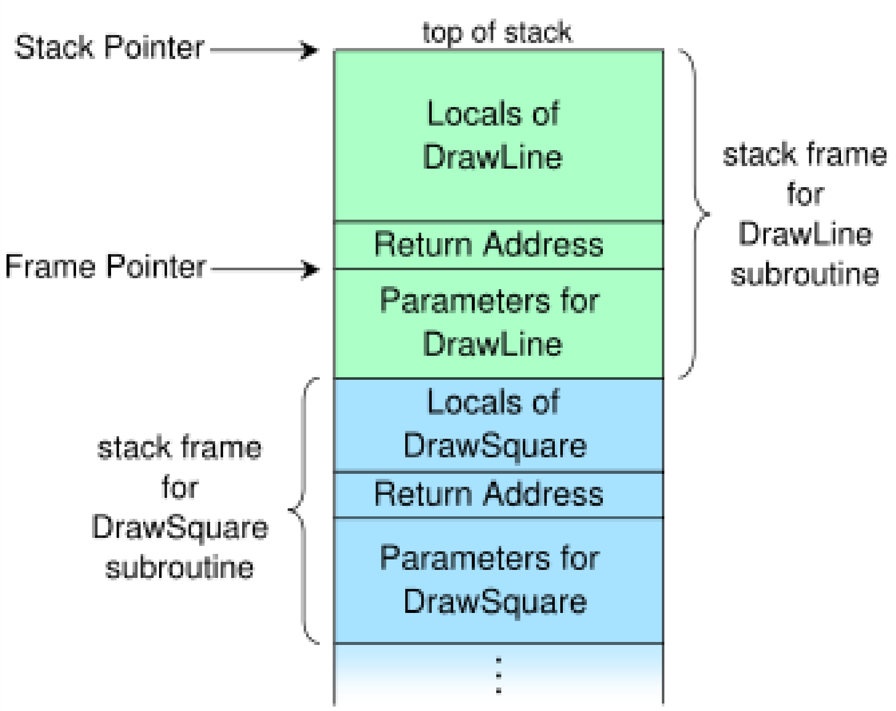
Local Variables at Method level

Block level local Variables



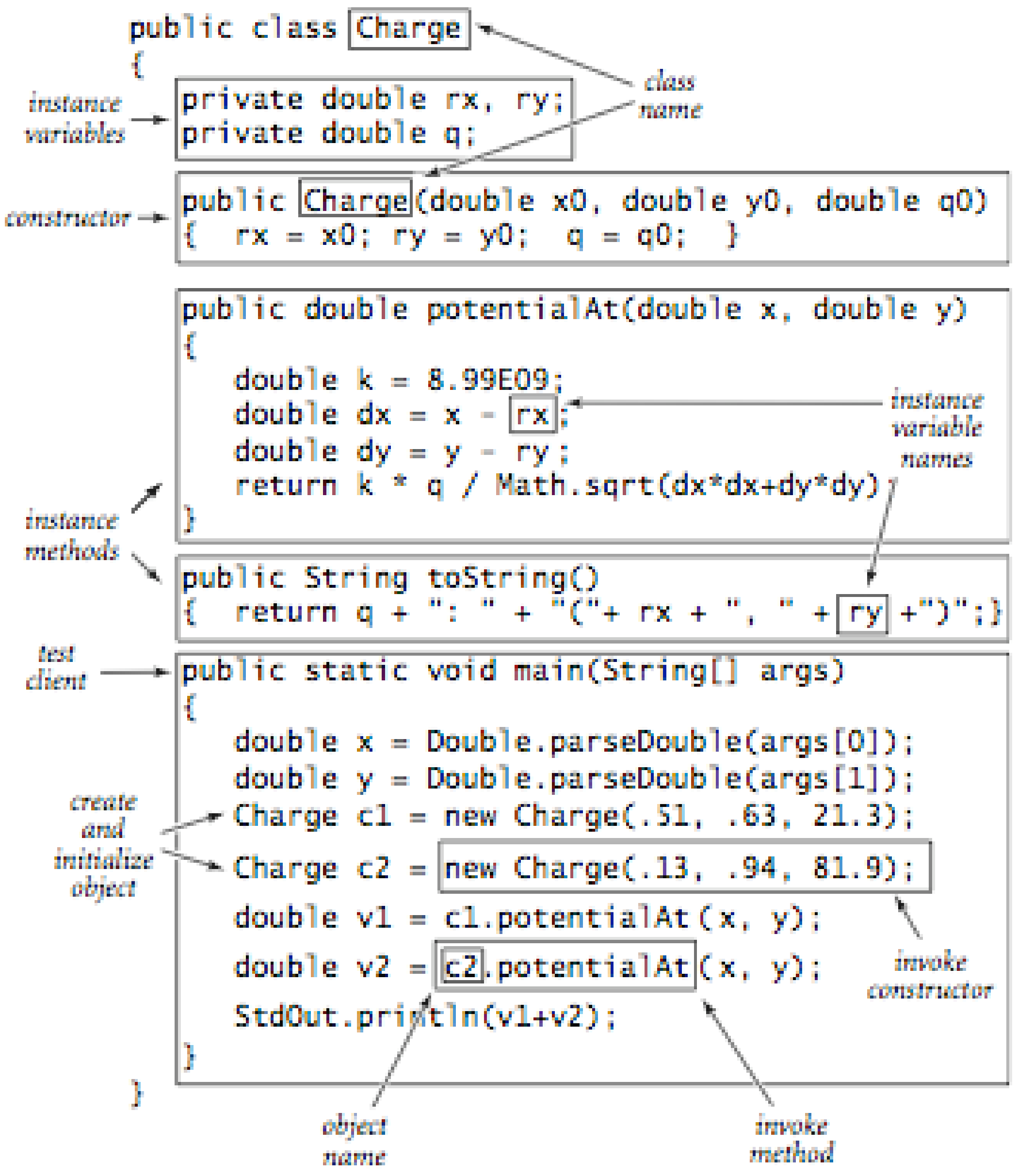
Scope of Local Variables and Parameters

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.



```
public class Charge
{
   private double rx, ry;
   private double q;

   public Charge(double x0, double y0, double q0)
   {  rx = x0; ry = y0;  q = q0;  }

   public double potentialAt(double x, double y)
   {
      double k = 8.99E09;
      double dx = x - rx;
      double dy = y - ry;
      return k * q / Math.sqrt(dx*dx+dy*dy);
   }

   public String toString()
   {  return q + ": " + "("+ rx + ", " + ry +")";}

   public static void main(String[] args)
   {
      double x = Double.parseDouble(args[0]);
      double y = Double.parseDouble(args[1]);
      Charge c1 = new Charge(.51, .63, 21.3);
      Charge c2 = new Charge(.13, .94, 81.9);
      double v1 = c1.potentialAt(x, y);
      double v2 = c2.potentialAt(x, y);
      StdOut.println(v1+v2);
   }
}
```

Anatomy of a class

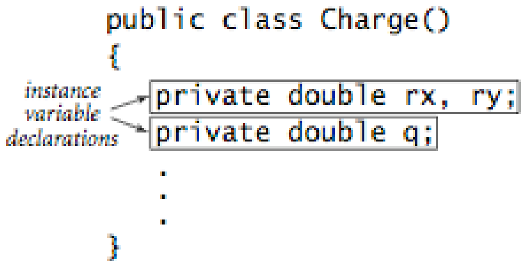
Instance Members

- 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.

Instance Variables



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

Instance variables

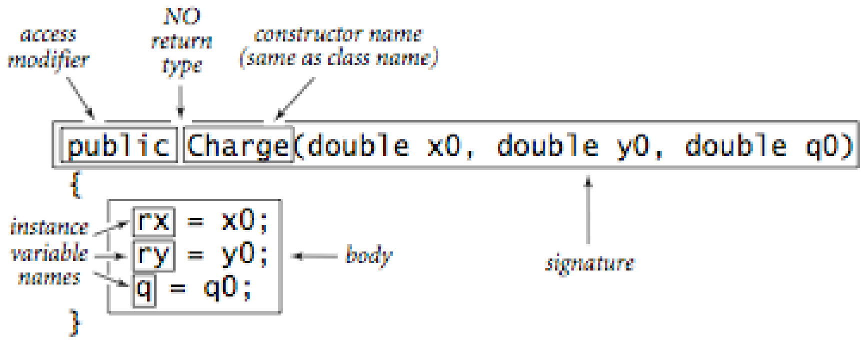
- To write code for the methods that manipulate data type values, the first thing that we need is to declare variables that we can use to refer to the values in code. These variables can be any type of data. We declare the types and names of these instance variables in the same way as we declare local 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 fun the corresp



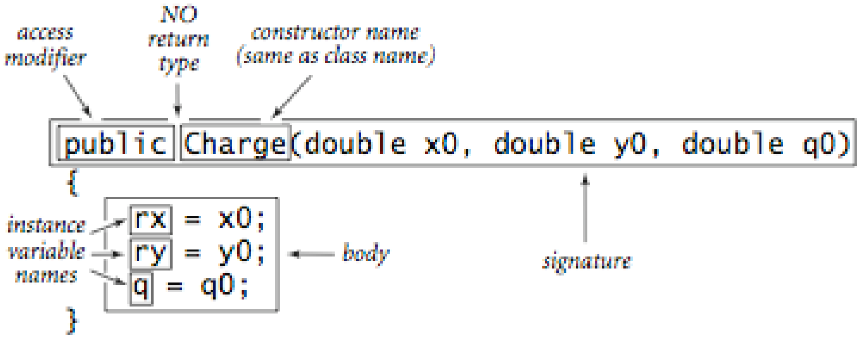
```
public Charge(double x0, double y0, double q0)
{
   rx = x0;
   ry = y0;
   q = q0;
}
```

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



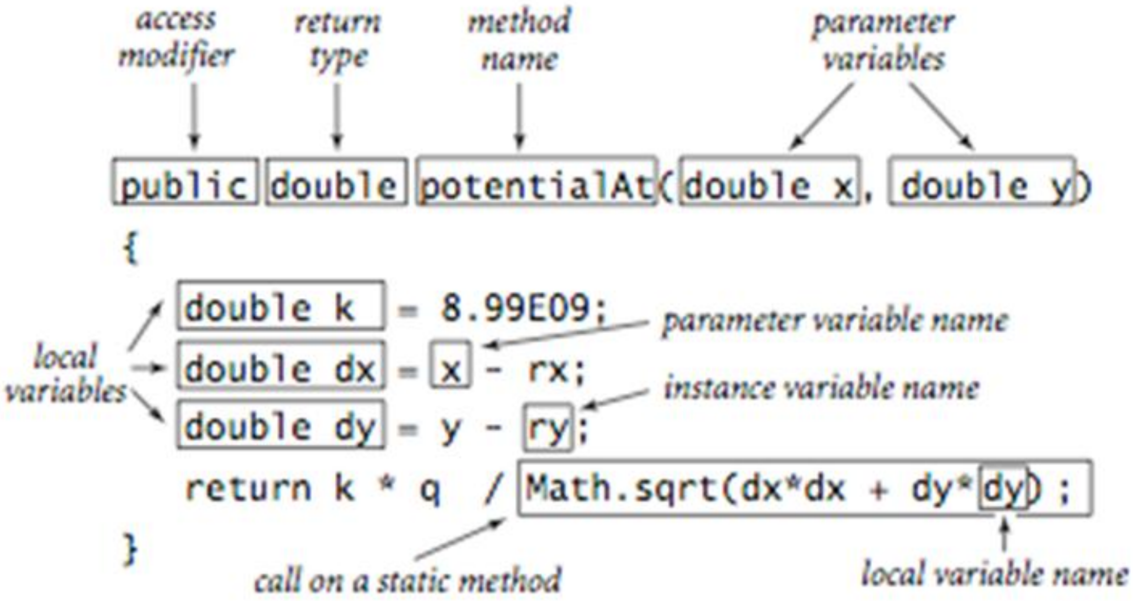
```
public Charge(double x0, double y0, double q0)
{
   rx = x0;
   ry = y0;
   q = q0;
}
```

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).



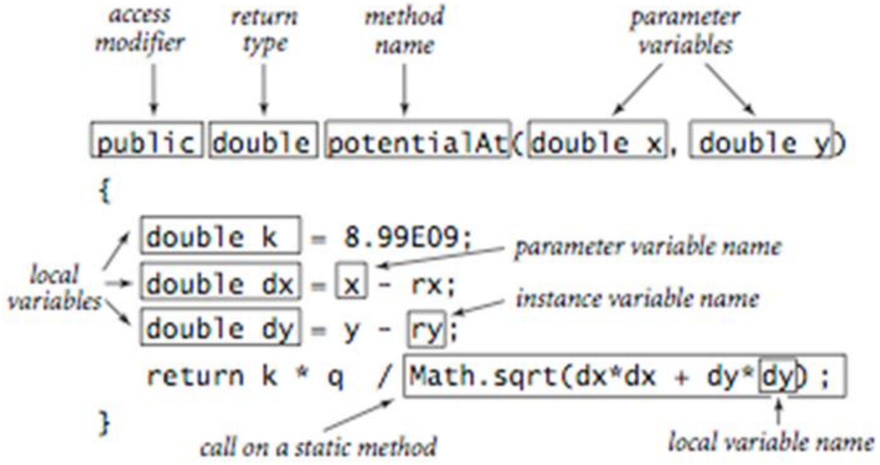
```
public double potentialAt(double x, double y)
{
   double k = 8.99E09;
   double dx = x - rx;
   double dy = y - ry;
   return k * q / Math.sqrt(dx*dx + dy*dy) ;
}
```

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.**



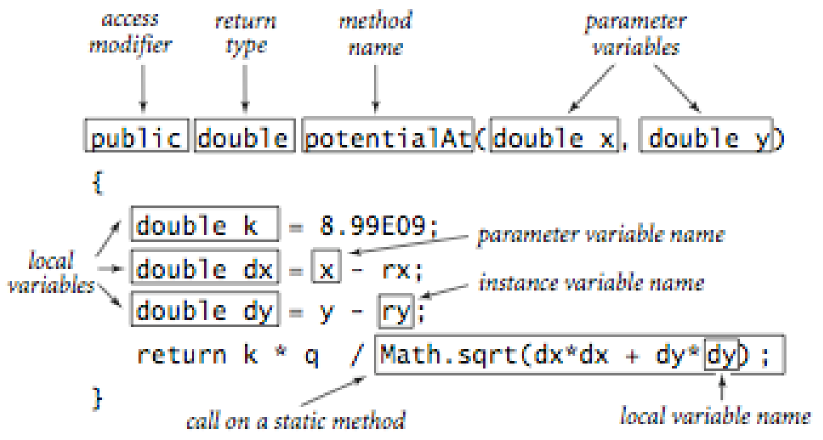
```
public double potentialAt(double x, double y)
{
   double k = 8.99E09;
   double dx = x - rx;
   double dy = y - ry;
   return k * q / Math.sqrt(dx*dx + dy*dy);
}
```

Anatomy of a data-type method

Instance methods

(methods in a class with static modifier)

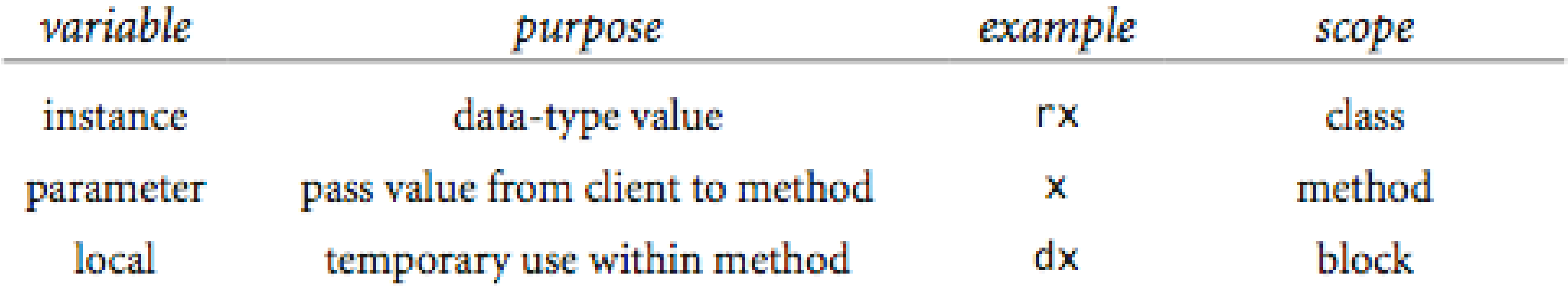
- When a client invokes a method, the parameter values are initialized with client values, the lines of code are executed until a return value is computed, and the value is returned to the client, with the same effect as if the method invocation in the client were replaced with that value. All of this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



```
public double potentialAt(double x, double y)
{
   double k  = 8.99E09;
   double dx = x - rx;
   double dy = y - ry;
   return k * q / Math.sqrt(dx*dx + dy*dy);
}
```

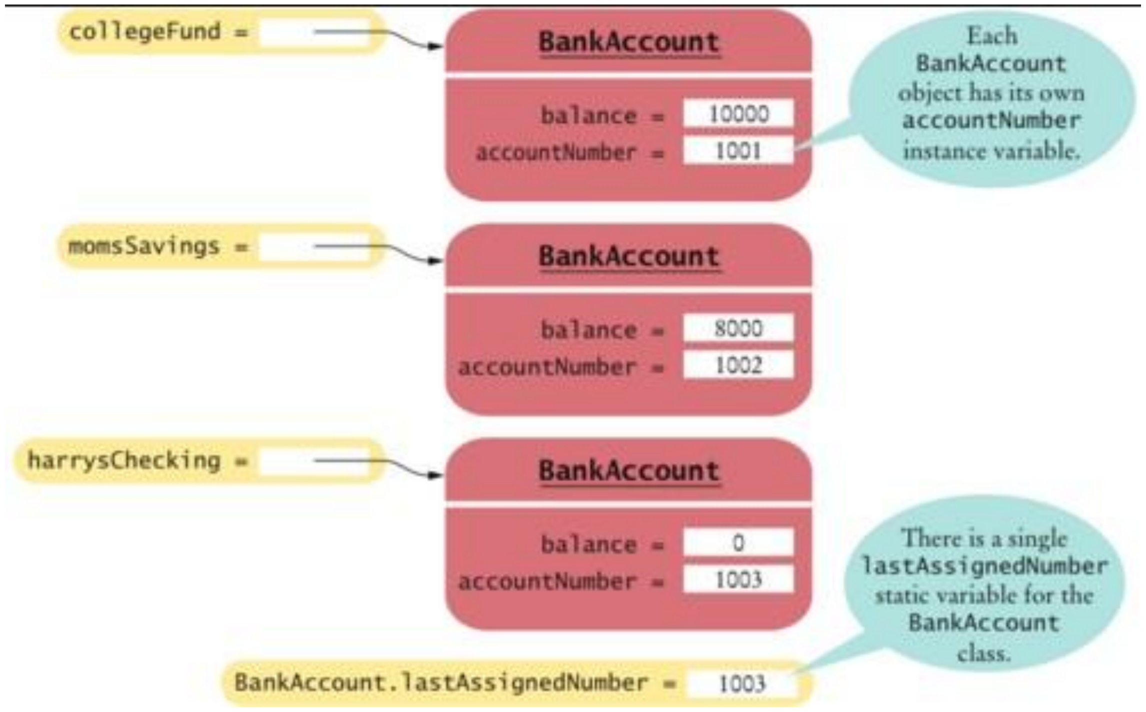
Anatomy of a data-type method

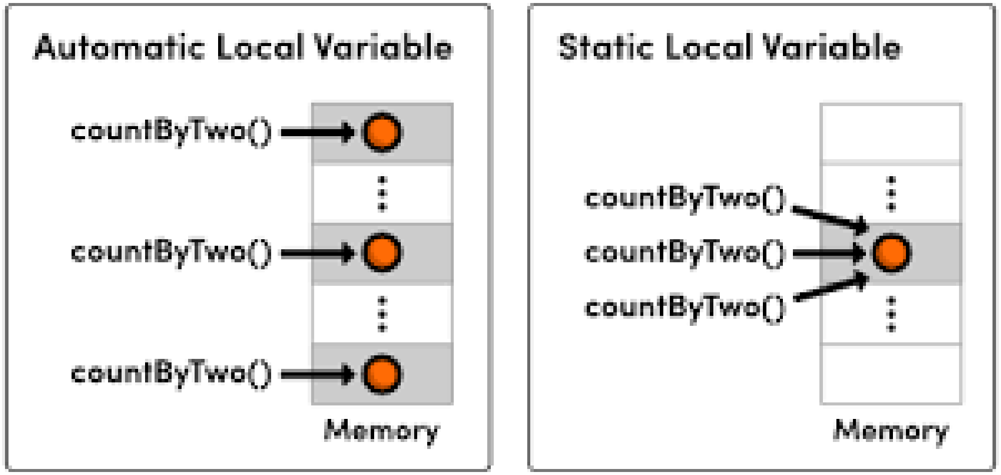
Summary of Local Variables



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

Static Members

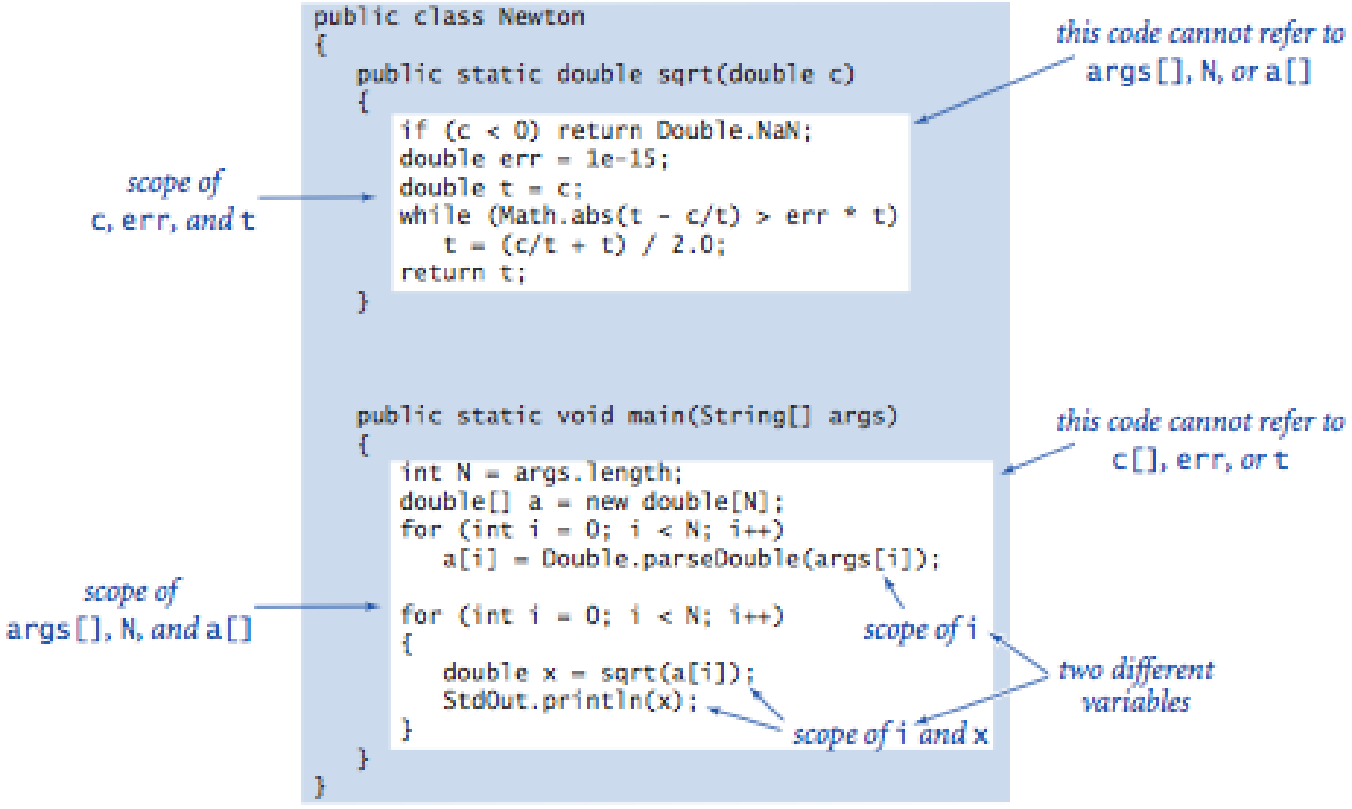

collegeFund =
BankAccount
balance = 10000
accountNumber = 1001
Each BankAccount object has its own accountNumber instance variable.
momsSavings =
BankAccount
balance = 8000
accountNumber = 1002
harrysChecking =
BankAccount
balance = 0
accountNumber = 1003
There is a single lastAssignedNumber static variable for the BankAccount class.
BankAccount.lastAssignedNumber = 1003



- *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)



```
public class Newton
{
   public static double sqrt(double c)
   {
      if (c < 0) return Double.NaN;
      double err = 1e-15;
      double t = c;
      while (Math.abs(t - c/t) > err * t)
         t = (c/t + t) / 2.0;
      return t;
   }


   public static void main(String[] args)
   {
      int N = args.length;
      double[] a = new double[N];
      for (int i = 0; i < N; i++)
         a[i] = Double.parseDouble(args[i]);

      for (int i = 0; i < N; i++)
      {
         double x = sqrt(a[i]);
         StdOut.println(x);
      }
   }
}
```

this code cannot refer to args[], N, or a[]
scope of c, err, and t
this code cannot refer to c[], err, or t
scope of args[], N, and a[]
scope of i
two different variables
scope of i and x

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 13

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;
  }
```

```
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;
  }
```

reference data type to current object

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 14

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.

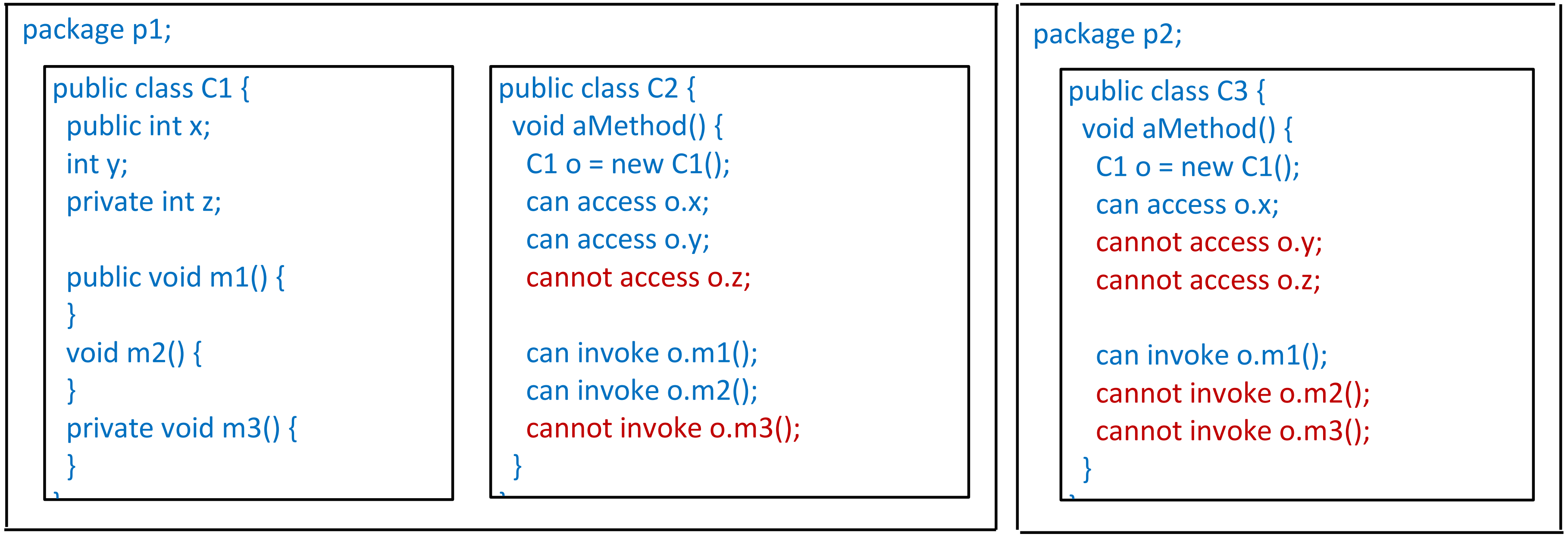
  private

  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

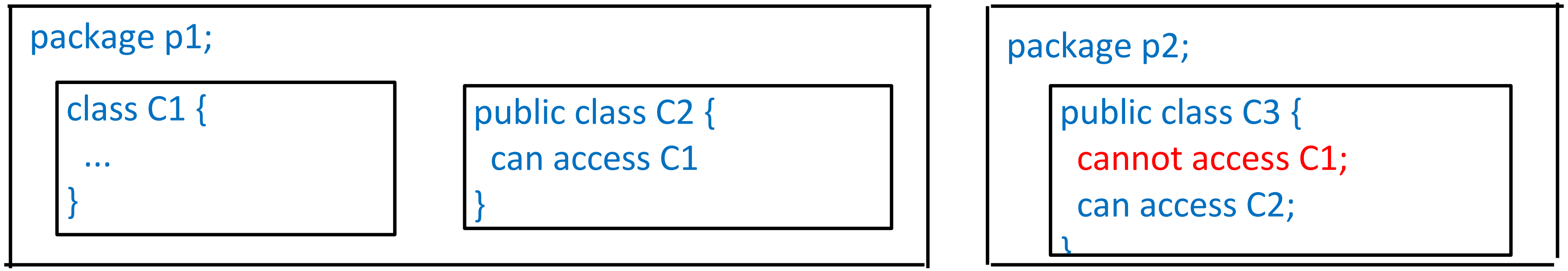
Modifier on Members in a class	Accessed from the same class	Accessed from the same package	Accessed from a subclass in a different package	Accessed from a different package
public	O	O	O	O
protected	O	O	O	X
default	O	O	X	X
private	O	X	X	X



```
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;
}
```

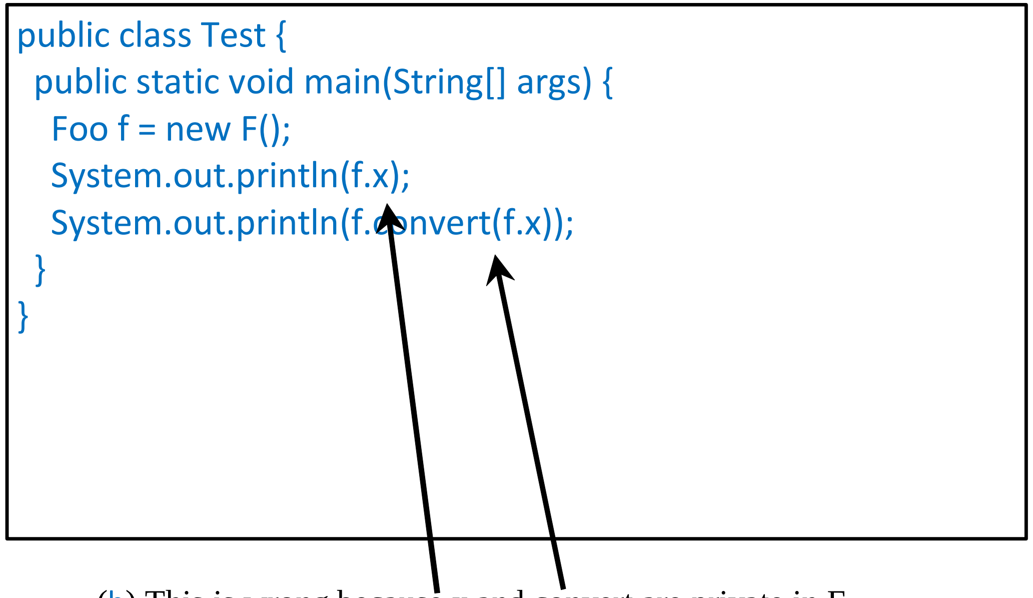
The private modifier restricts access to within a class, the default modifier restricts access to within a package, and the public modifier enables unrestricted access.

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.