AP Computer Science B

Java Object-Oriented Programming [Ver. 3.0]

Unit 4: Object-Oriented Programming

CHAPTER 10B: DATA ABSTRACTION

DR. ERIC CHOU IEEE SENIOR MEMBER

Objectives

- Class Design Style
- Data Classes From Library
- Classes in AP CSA Exam
- Class Data Encapsulation: Data Abstraction, Passing Object to Methods and Immutable Class
- **this** Reference
- Array of Objects, ArrayList of Objects
- Objects with Arrays, Objects with ArrayList

Class Design Style

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 getEngishScore() {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).

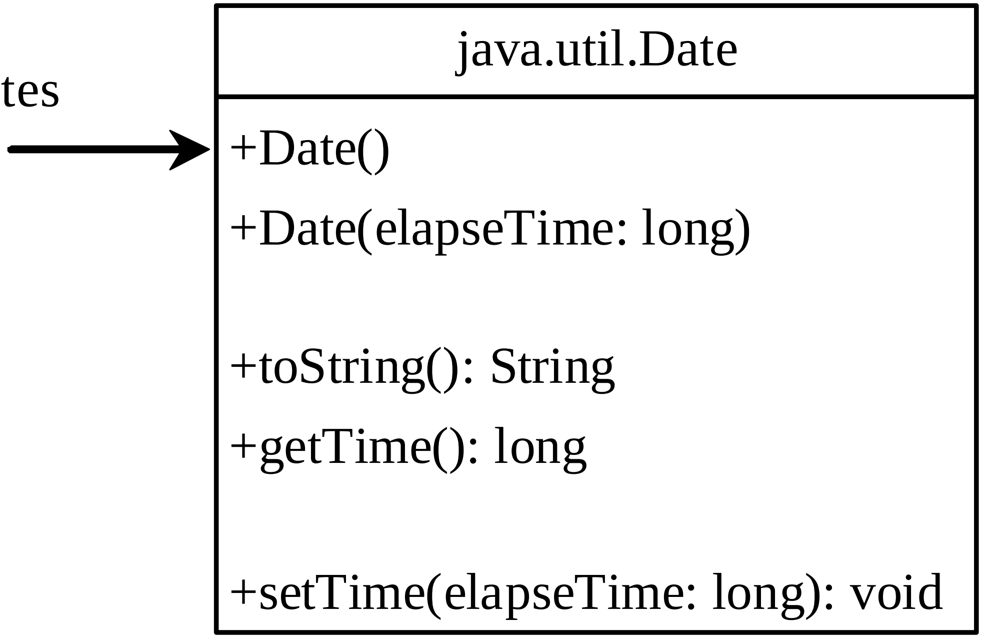
Data Classes From Java Library

LECTURE 2

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

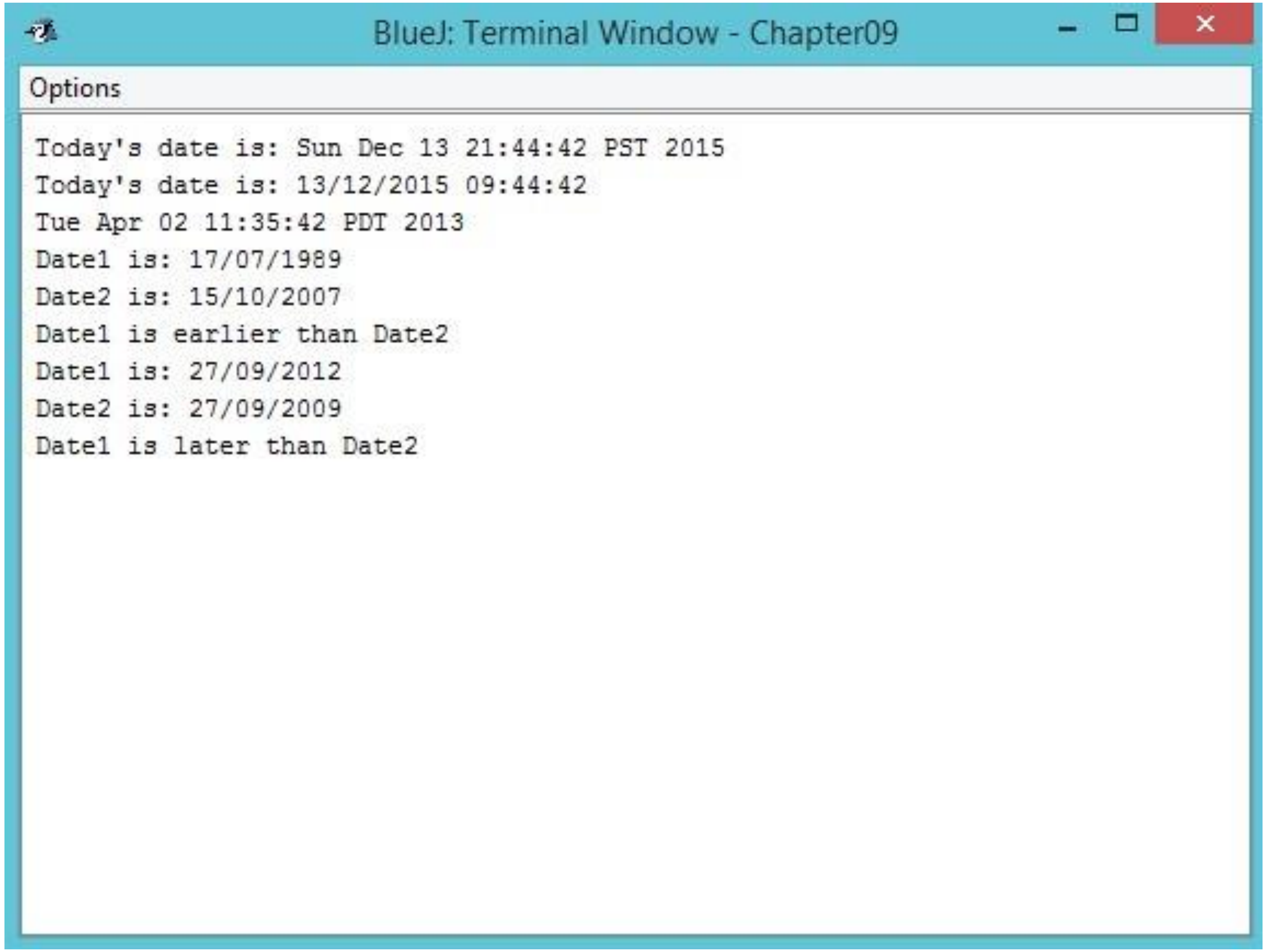
```
java.util.Date date = new
  java.util.Date();
System.out.println(date.toString());
```

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

Go BlueJ.

Demonstration Program

DATEEXAMPLE.JAVA



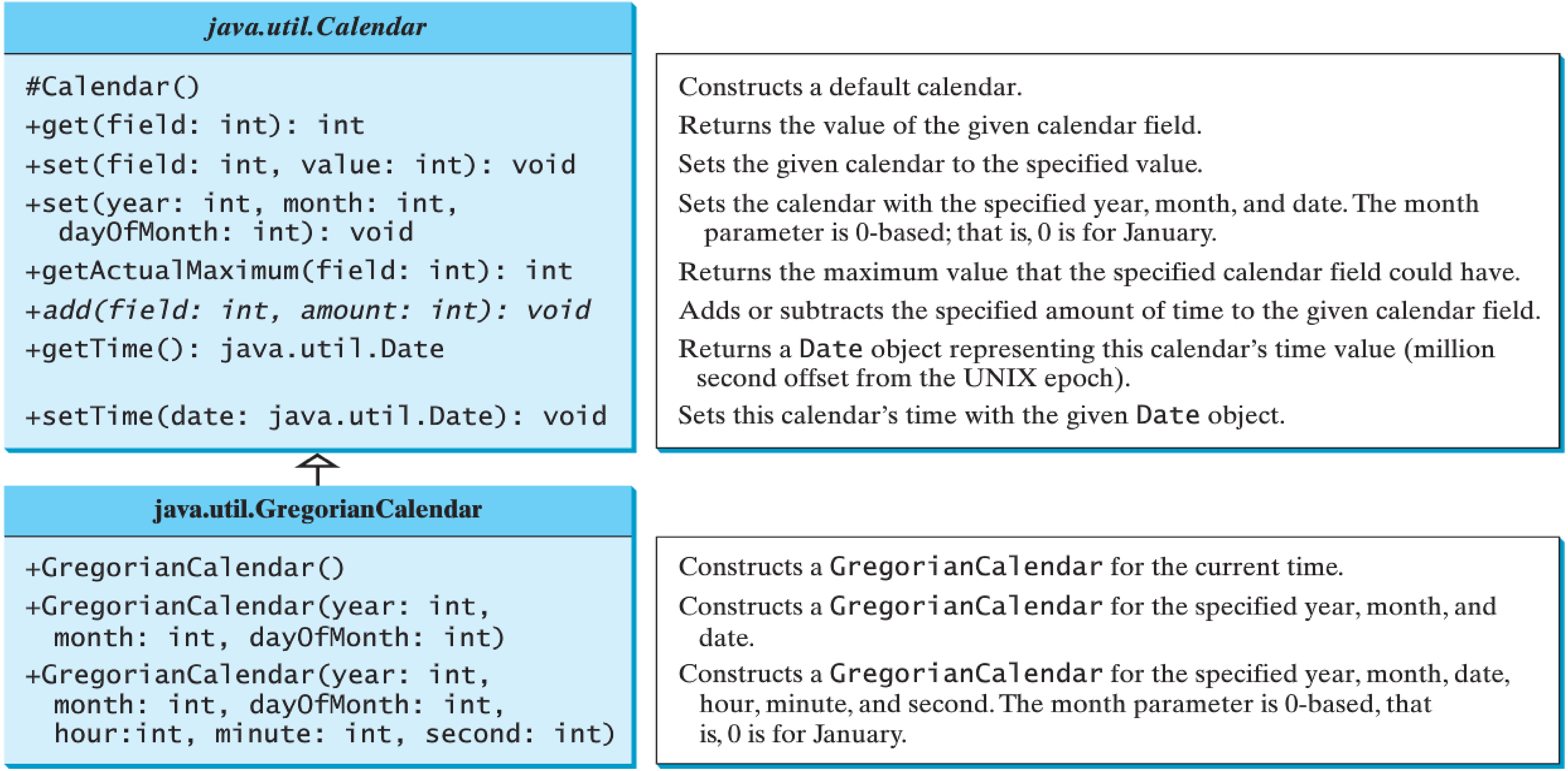
```
Today's date is: Sun Dec 13 21:44:42 PST 2015
Today's date is: 13/12/2015 09:44:42
Tue Apr 02 11:35:42 PDT 2013
Date1 is: 17/07/1989
Date2 is: 15/10/2007
Date1 is earlier than Date2
Date1 is: 27/09/2012
Date2 is: 27/09/2009
Date1 is later than Date2
```

Result: DateExample.java

Calendar Class

Provide date information in certain calendar format.

Go BlueJ !!!



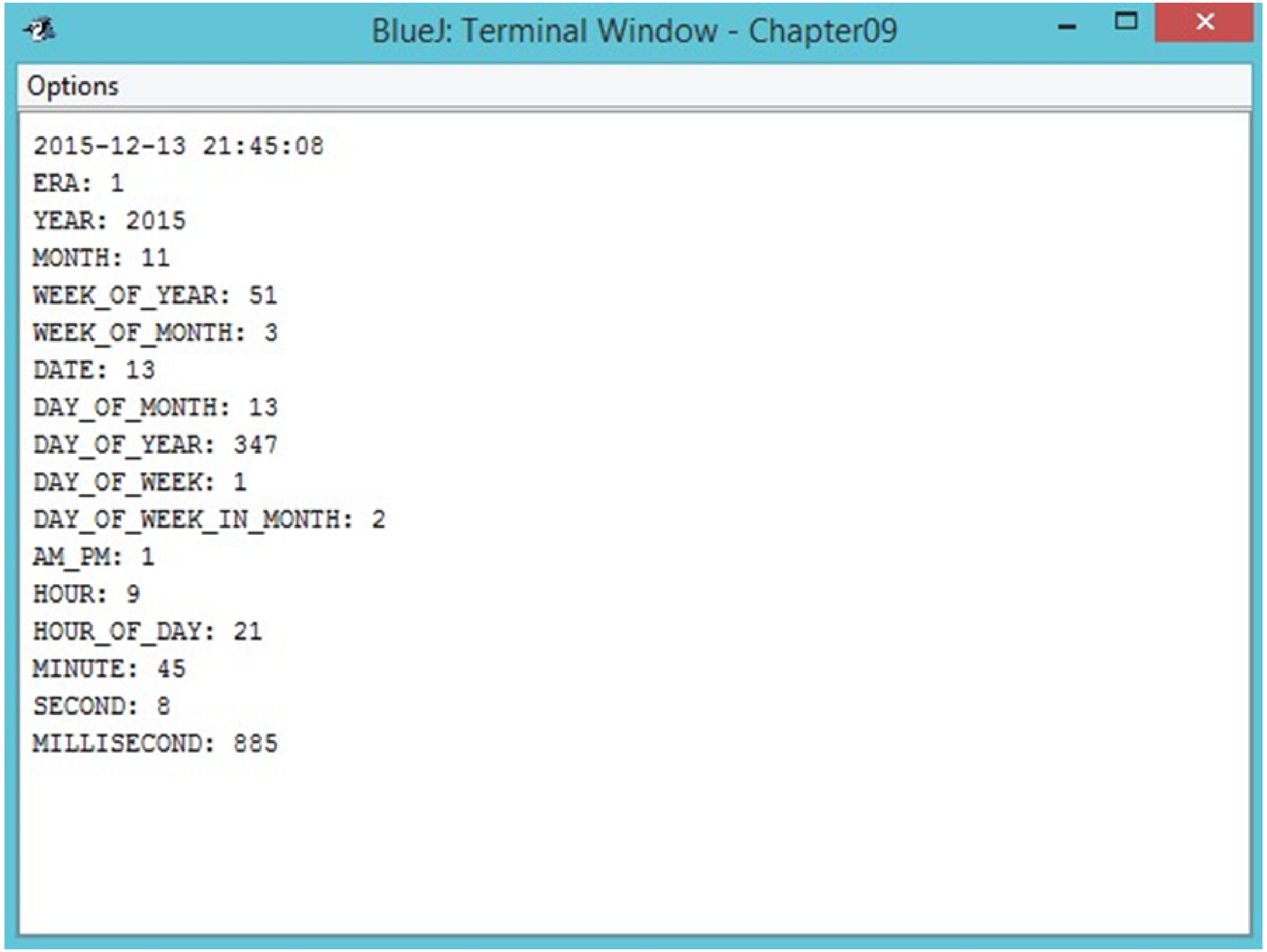
java.util.Calendar	
`#Calendar()`	Constructs a default calendar.
`+get(field: int): int`	Returns the value of the given calendar field.
`+set(field: int, value: int): void`	Sets the given calendar to the specified value.
`+set(year: int, month: int, dayOfMonth: int): void`	Sets the calendar with the specified year, month, and date. The month parameter is 0-based; that is, 0 is for January.
`+getActualMaximum(field: int): int`	Returns the maximum value that the specified calendar field could have.
`+add(field: int, amount: int): void`	Adds or subtracts the specified amount of time to the given calendar field.
`+getTime(): java.util.Date`	Returns a `Date` object representing this calendar's time value (million second offset from the UNIX epoch).
`+setTime(date: java.util.Date): void`	Sets this calendar's time with the given `Date` object.

java.util.GregorianCalendar	
`+GregorianCalendar()`	Constructs a `GregorianCalendar` for the current time.
`+GregorianCalendar(year: int, month: int, dayOfMonth: int)`	Constructs a `GregorianCalendar` for the specified year, month, and date.
`+GregorianCalendar(year: int, month: int, dayOfMonth: int, hour:int, minute: int, second: int)`	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.

CalendarExample.java

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

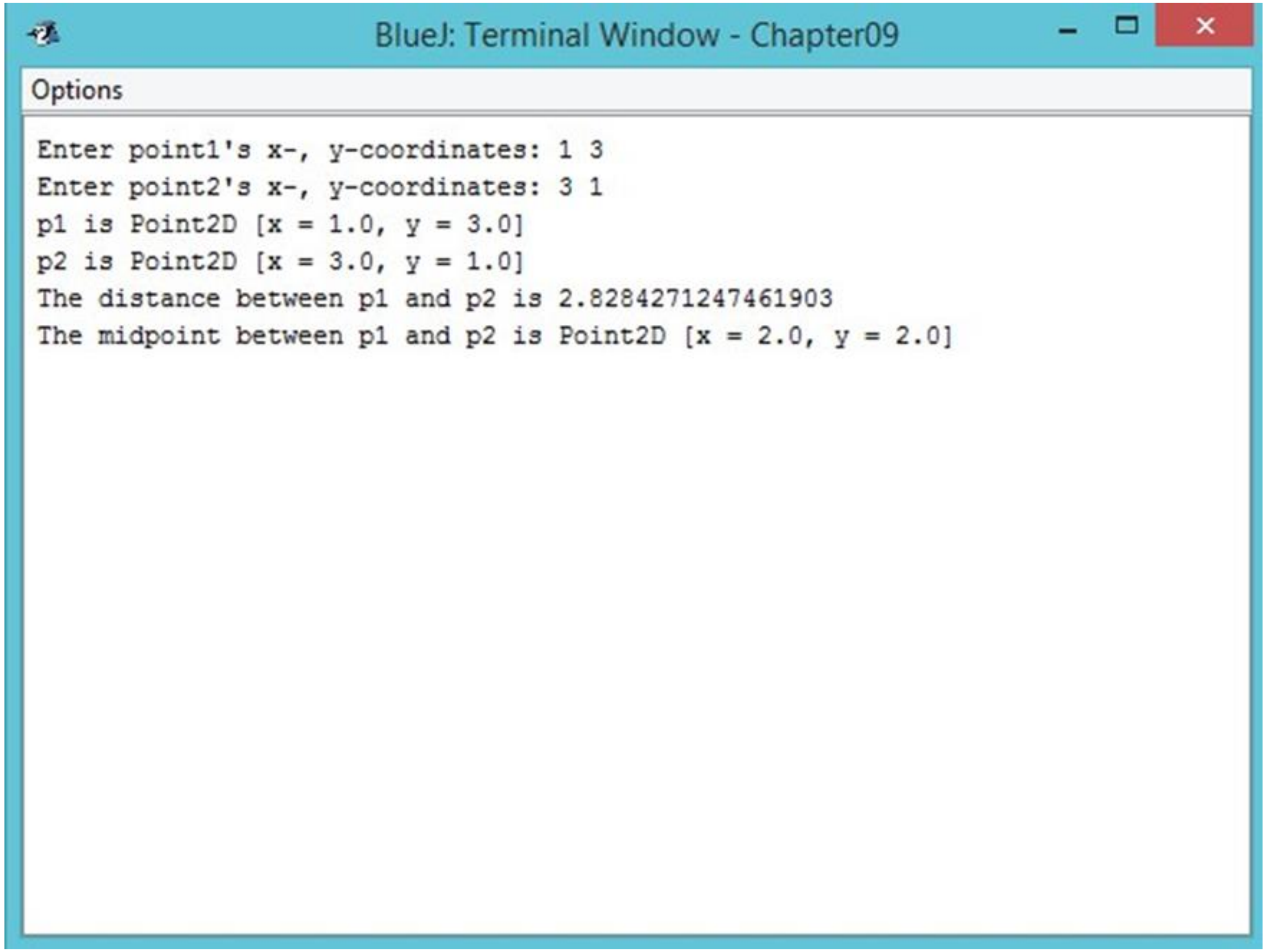
Point2D Class

- 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 3

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

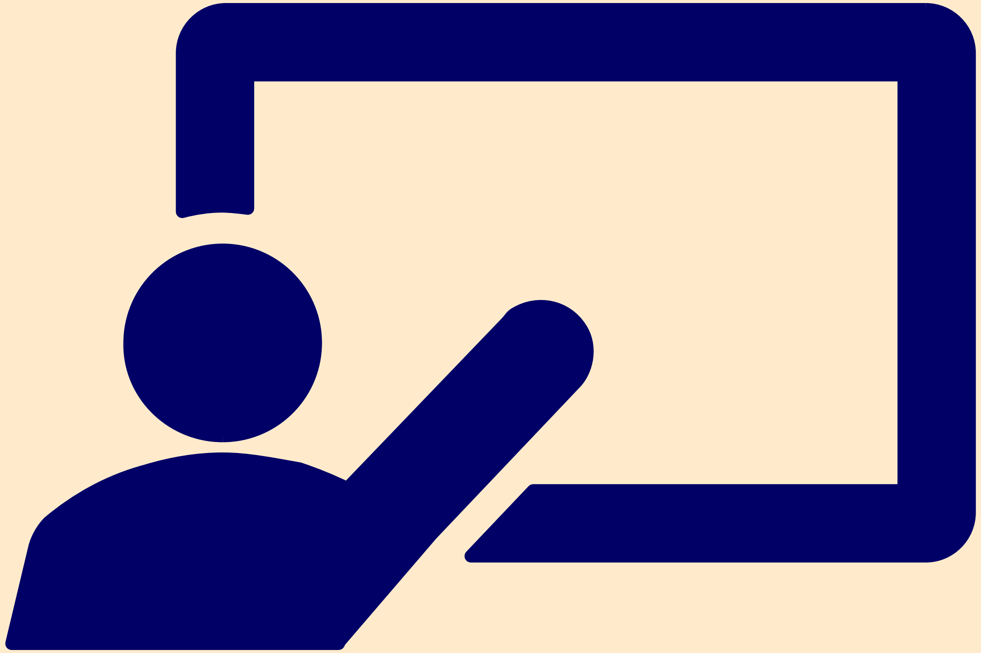
Date/Time: Not Covered

Graphics/Geometry: Note Covered

Image: GUI

Video: GUI

Audio: GUI

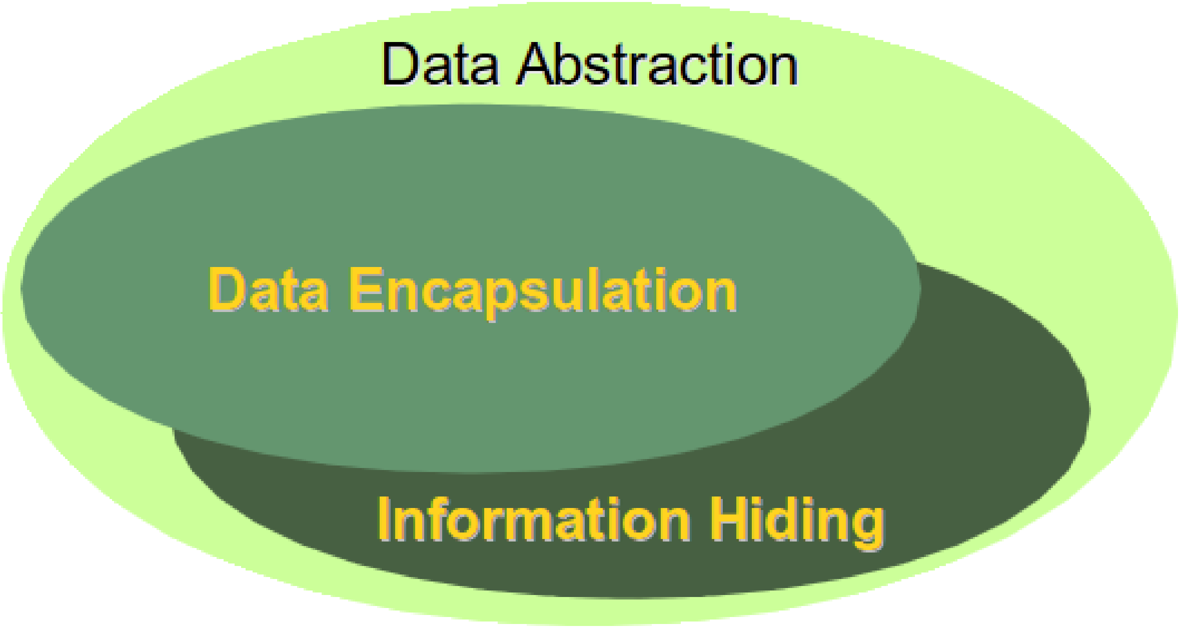


Data Abstraction

LECTURE 4

Data Field Encapsulation
Why Data Fields Should Be private?

- To protect data.
- To make class easy to maintain.



Class/Methods are also considered abstraction.

Constants, Static Variables, … Overloading are also considered as Information hiding.

What is Encapsulation

Private Data Fields and Public Accessors and Mutators

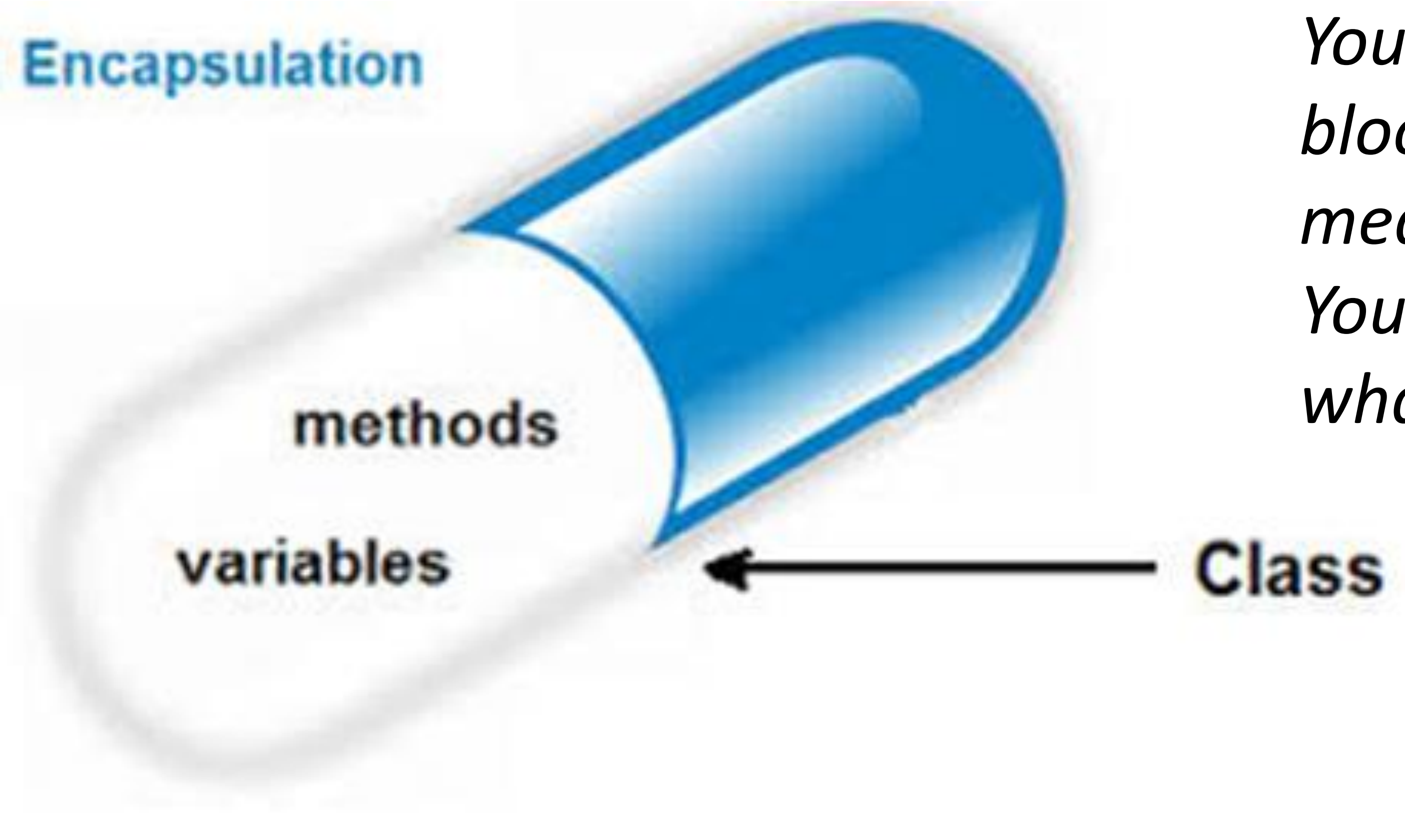
- The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class. However if we setup public getter and setter methods to update (for e.g. void setSSN(int ssn))and *read (for e.g. int getSSN()),* the private data fields.
- Then, the outside class can access those private data fields via public methods.

What is Encapsulation

Private Data Fields and Public Accessors and Mutators

- **This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes.** That’s why encapsulation is known as **data hiding**. We will see an example to understand this concept later.

Data Field Encapsulation
Why Data Fields Should Be private?



You just need to know it is blood pressure control medicine.
You do not need to know what chemical formula it is.

Visibility Modifiers

	`public`	`private`
Variables	Violate encapsulation	Enforce encapsulation
Methods	Provide services to clients	Support other methods in the class

Data Encapsulation Topics

Data Encapsulation: All Private Data Fields and Public Accessors and Mutators. (**data hiding**, **information hiding**: first Object-Oriented feature discussed so far.)

How to Enter Data Capsules (Accessing Objects):

(1) Using Public Mutators

(2) Passing Objects to Methods.

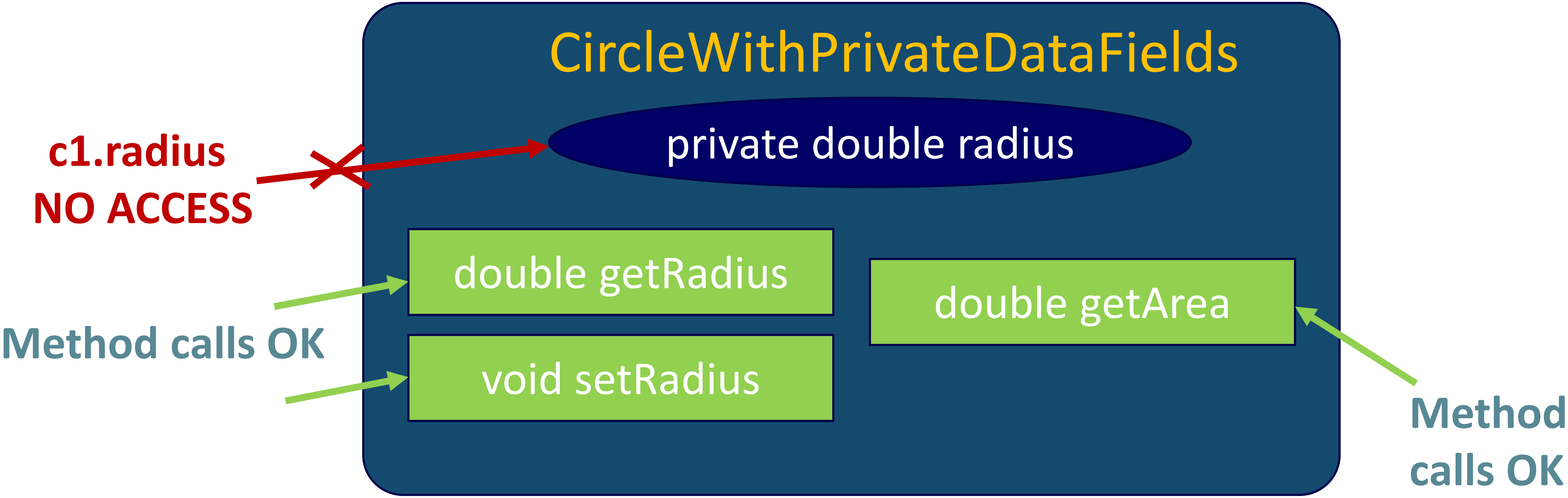
Data Encapsulation Topics

Immutable Classes (Object): All Private Data Fields, No Mutators and No Accessor Method Returning Reference Data.

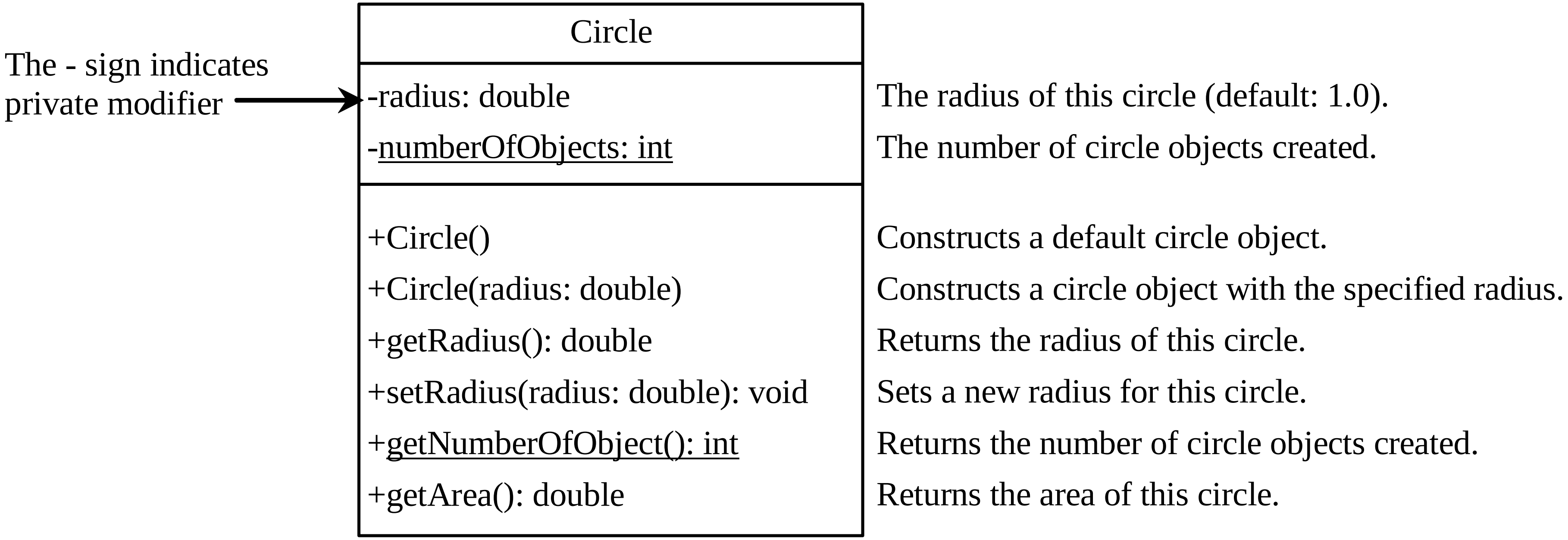
Data Encapsulation

LECTURE 5

Encapsulation Using Private Data Fields



Example of
Data Field Encapsulation



The - sign indicates private modifier

Circle

-radius: double	The radius of this circle (default: 1.0).
-numberOfObjects: int	The number of circle objects created.
+Circle()	Constructs a default circle object.
+Circle(radius: double)	Constructs a circle object with the specified radius.
+getRadius(): double	Returns the radius of this circle.
+setRadius(radius: double): void	Sets a new radius for this circle.
+getNumberOfObject(): int	Returns the number of circle objects created.
+getArea(): double	Returns the area of this circle.

CircleWithPrivateDataFields

```
Class Header (Body Omitted)
  public class CircleWithPrivateDataFields {
  private double radius;
  private static int numberOfObjects;
  public CircleWithPrivateDataFields(){..}
  public CircleWithPrivateDataFields(double newRadius){..}
  public double getRadius(){..}
  public void setRadius(double newRadius) {..}
  public static int getNumberOfObjects(){..}
  public double getArea(){..}
}
```

Static Members (Class Members)
Properties
- static int numberOfObjects
Methods
+ static int getNumberOfObjects()
Non-static Members (Instance Members)
Properties
- double radius
Constructors
+ CircleWithStaticMembers()
+ CircleWithStaticMemebrs(double newRadius)
Methods
+ double getRadius()
+ void setRadius()
+ double getArea()

Demonstration Program

CIRCLEWITHPRIVATEDATAFIELDS.JAVA

TESTCIRCLEWITHPRIVATEDATAFIELDS.JAVA

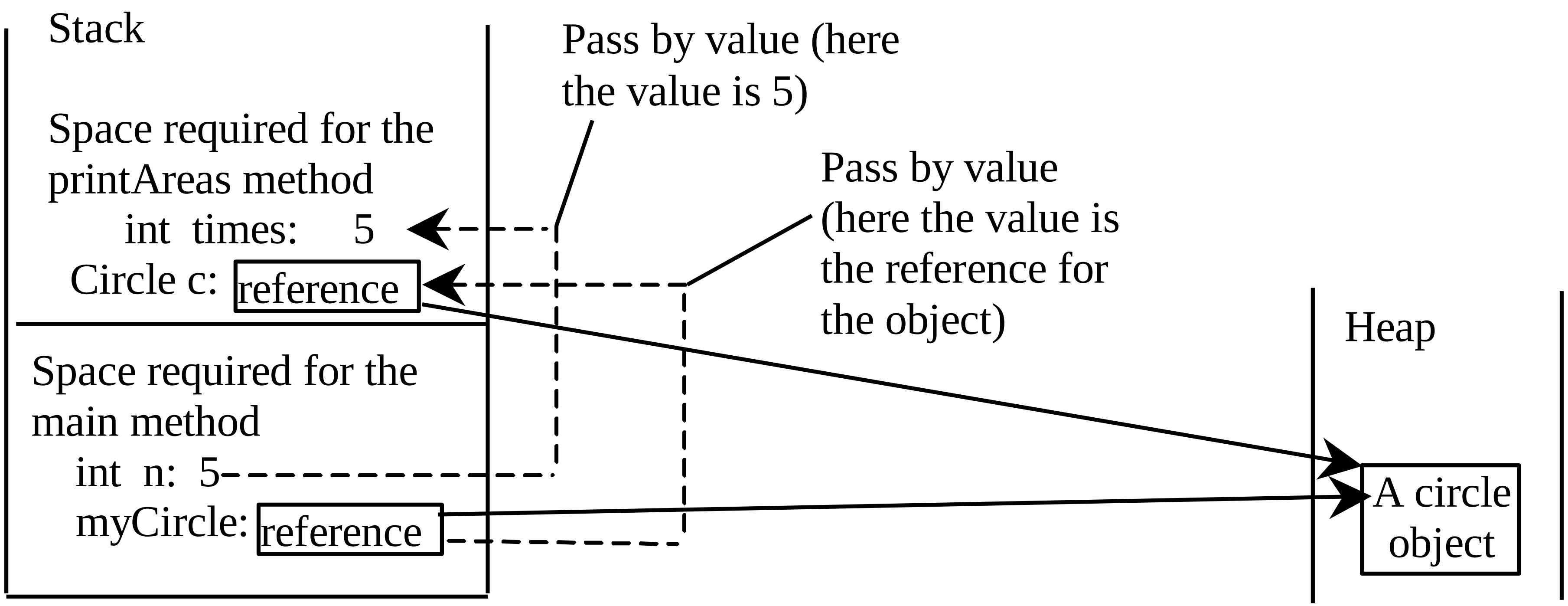
Passing Objects to Methods

LECTURE 6

Passing Objects to Methods

- Passing by value for primitive type value (the value is passed to the parameter)
- Passing by value for reference type value (the value is the reference to the object)

Passing Objects to Methods, cont.


Stack
Space required for the printAreas method
int times: 5
Circle c: reference
Space required for the main method
int n: 5
myCircle: reference
Pass by value (here the value is 5)
Pass by value (here the value is the reference for the object)
Heap
A circle object

Passing Objects is Another Way to Enter into **Data Capsule (Object)**

Reference data type pointing to the body of the object. So, when the method get a argument of reference type, the method only get the pointer.

The pointer's accessing power enables the method's other code to access and modify the data in the **Data Capsule (Encapsulated Objects)**

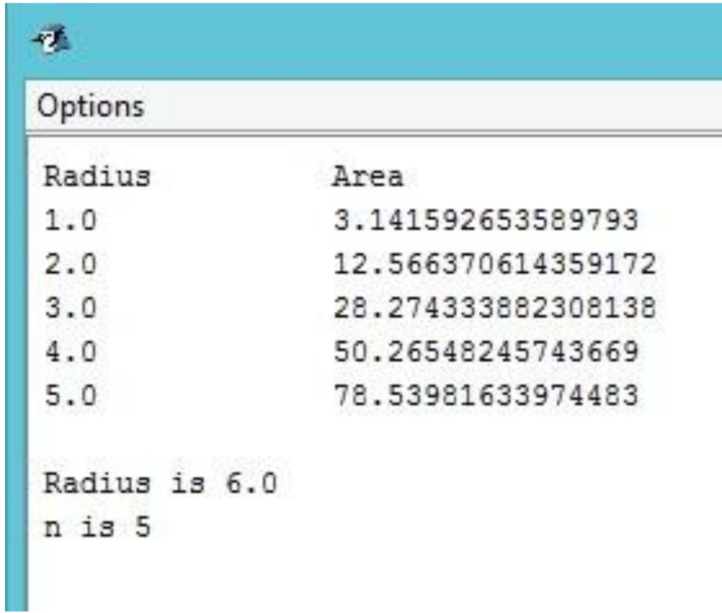
Passing Objects is an action of opening data capsule.

Pointer View for Passing Objects to Methods

PassingObjects.java

```
public static void main(String[] args) {
  // Create a Circle object with radius 1
  CircleWithPrivateDataFields myCircle =
          new CircleWithPrivateDataFields(1);
  // Create a circle with radius 1.0

  // Print areas for radius 1, 2, 3, 4, and 5.
  int n = 5;
  printAreas(myCircle, n);

  // See myCircle.radius and times
  System.out.println("\n" + "Radius is "
                   + myCircle.getRadius());
  System.out.println("n is " + n);
}
```

```
public static void printAreas(CircleWithPrivateDataFields c,
int times) {
  System.out.println("Radius \t\tArea");
  while (times >= 1) {
    System.out.println(c.getRadius() + "\t\t" + c.getArea());
    c.setRadius(c.getRadius() + 1);
    times--;
  }
}
```



Options

```
Radius          Area
1.0             3.141592653589793
2.0             12.566370614359172
3.0             28.274333882308138
4.0             50.26548245743669
5.0             78.53981633974483

Radius is 6.0
n is 5
```

Demonstration Program

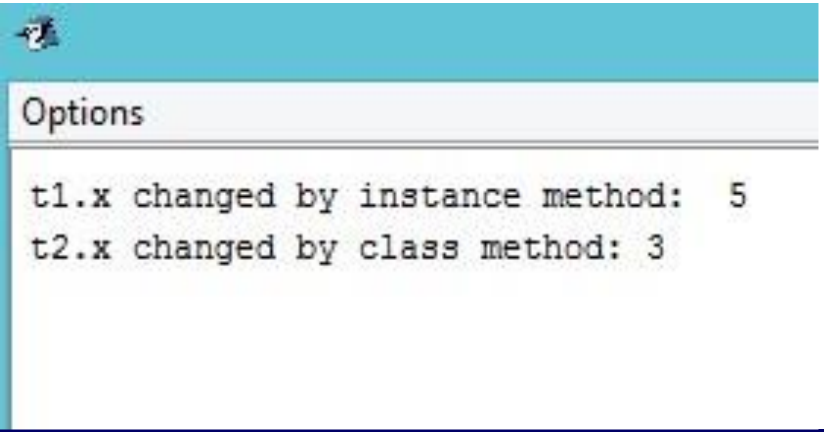
PASSINGOBJECTS.JAVA

Two Ways to Update a Data Field from Other Class

INT2.java TestINT2.java

```
class TestINT2{
    public static void main(){
      INT2 t1 = new INT2();
      INT2 t2 = new INT2();
      System.out.println
        (“t1.x changed by instance method:  ”+t1.change());
      System.out.println
        (“t2.x changed by class method: ” + INT2.change(t2));
   }
}
```

```
class INT2{
  private int x=0;
  public int change(){
         x = 5;
         return x;
  }
   public static int change(INT2 a){
         a.x = 3; // can not be x (nonstatic)
         return a.x;
   }
   // static can work on nonstatic instance data
   // if you pass object.
}
```



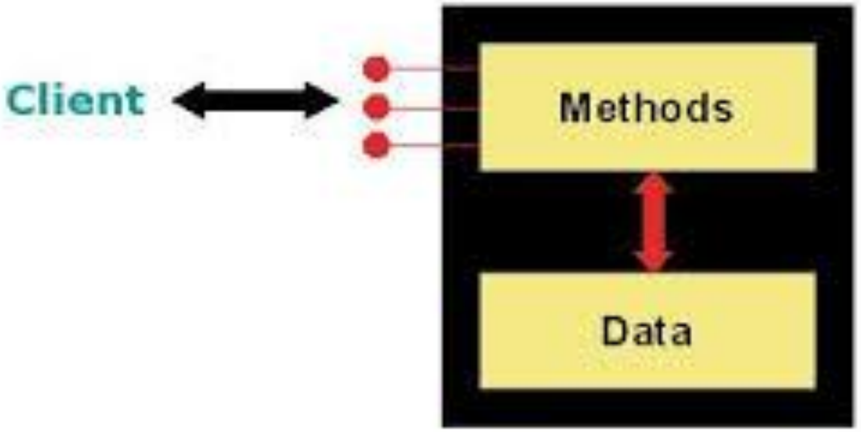
```
t1.x changed by instance method:  5
t2.x changed by class method: 3
```

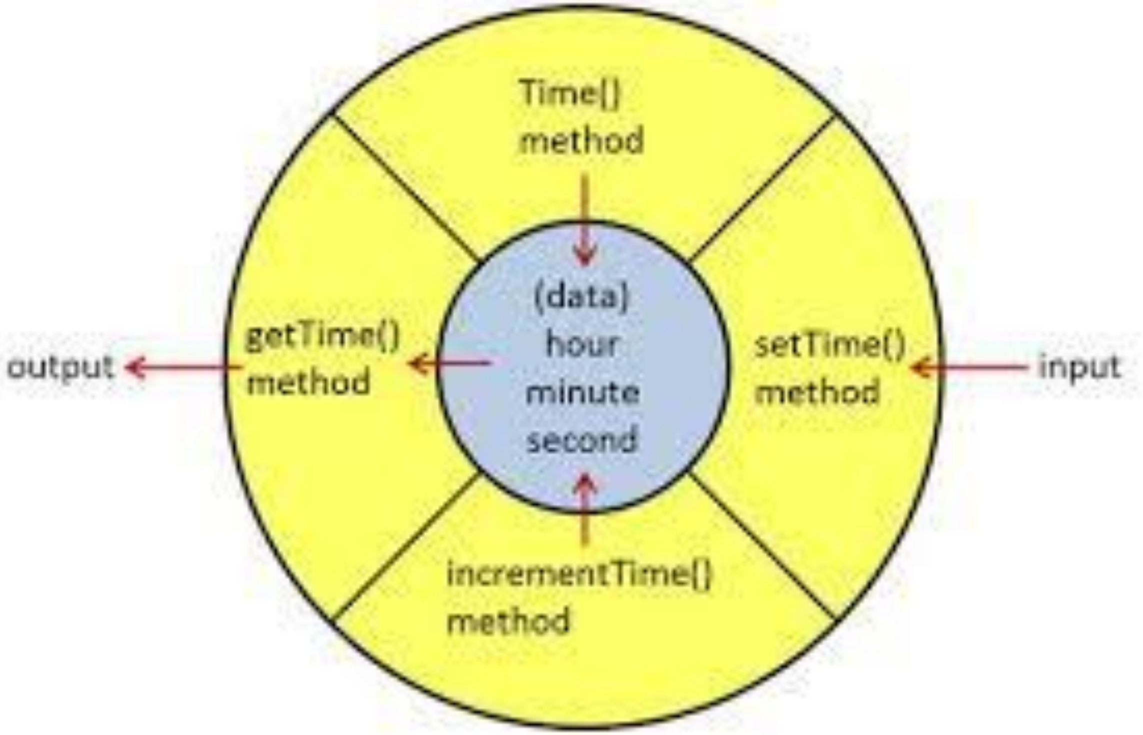
Demonstration Program

INT2.JAVA + TESTINT2.JAVA

First Way: Accessor/Mutator Methods

For Encapsulated Objects





Immutable Objects and Classes

LECTURE 7

Immutable Objects and Classes

You can define immutable classes to create immutable objects. The contents of immutable objects cannot be changed.

- The String class is immutable.
- If a class is immutable, then all its data fields must be private and it cannot contain public setter methods for any data fields. A Class with **all private data fields** and **no mutators** is not necessarily immutable. For example, the following Student class has all private data fields and no setter methods,

Immutable Objects and Classes

```
public class Student {
   private int id;
   private String name;
   private java.util.Date dateCreated;
   public Student(int ssn, String newName) {
        id = ssn;
        name = newName;
        dateCreated = new java.util.Date();
   }
   public int getId() {  return id;   }
   public String getName() {return name; }
   public java.util.Date getDateCreated() {
       return dateCreated;
   }
}
```

Immutable Objects and Classes

- As shown in the following code, the data field dateCreated is returned using the getDateCreated() method. This is a reference to a Date object. Through this reference, the content for dateCreated can be changed.

```
public class Test{
    public static void main(String[] args){
        Student student = new Student(111223333, “John”);
        java.util.Date dateCreated = student.getDateCreated();
        dateCreated.setTime(200000);
    }
}
```

// Similar to shallow copy, you make the reference data type private, but not its body. So, leave a door open to change of contents.

// It is no longer immutable.

Immutable Objects and Classes

- For a class to be immutable, it must meet the following requirements:
- All **data fields** must be **private**.
- There can’t be any mutator methods for data fields. (**No Mutator**)
- No accessor methods can return a reference to data field that is mutable. (**No accessor method for reference data.)**

this Reference

LECTURE 8

The this Reference

- The keyword this refers to the object itself. It can also be used inside a constructor to invoke another constructor of the same class, when the constructor is overloaded.

- The *this* keyword is the name of a reference that an object can use to refer to itself. You can use the *this* keyword to reference the object's instance members. For example, the following code in (a) uses *this* to reference the object's radius and invokes its *getArea()* method explicitly. The *this* reference is normally omitted, as shown in (b). However, the *this* reference is needed to reference hidden data fields or invoke an overloaded constructor.

The this Reference

```
public class Circle {
  private double radius;
  …
  public double getArea(){
    return this.radius * this.radius * Math.PI;
  }
  public String toString(){
     return “radius: ” + this.radius + “area: ” + this.getArea();
 }
}
```

(a)

(a) is equivalent to (b)

```
public class Circle {
  private double radius;
  …
  public double getArea(){
    return radius * radius * Math.PI;
  }
  public String toString(){
     return “radius: ” + radius + “area: ” + getArea();
 }
}
```

(b)

Using this to Reference Hidden Data Fields

- When there is a local variable or a parameter sharing the same name with a class variable, we can use this.variable to refer to the instance variable (object variable) while the variable is used for the local or parameter variable. The instance variable is in fact hidden data field according to the rule for variable scope.
- A hidden static variable can be accessed simply by using the ClassName.staticVariable. A hidden instance variable can be accessed by using the keyword this.

Using this to Reference Hidden Data Fields

class variable(Class.var), instance variable (this.var)

```
public class F {
   private int i = 5;
   private static double k = 0;
   public void setI(int i) { this.i = i; }
   public static void setK(double k) {
       F.k = k;
   }
}
```

(a)

Suppose that f1 and f2 are two objects of F.

Invoking f1.setI(10) is to execute this.i = 10, where this refers f1

Invoking f2.setI(45) is to execute this.i = 45, where this refers f2

Invoking F.setK(33) is to execute F.k = 33. setK is a static method

The this Reference

- The **this** keyword gives us a way to reference the object that invokes an instance method. To invoke **f1.setI(10), this.i = i** is executed, which assigns the value of parameter i to the data field **i** of this calling object **f1**.
- The keyword **this** refers to the object that invokes the instance method setI. The line **F.k = k** means that the value in parameter **k** is assigned to the static data field **k** of the class, which is shared by all objects of the class.

Using this to Invoke a Constructor

- The *this* keyword can be used to invoke another constructor of the same class. For example, you can rewrite the Circle class as follows:

```
public class Circle {
   private double radius;
   public Circle(double radius) {
     // The this keyword is used to reference
     // the hidden data field radius
     this.radius = radius;
}
// of the object being constructed.
public Circle() {
 // The this keyword is used to invoke another constructor.
   this(1.0);
}
}
```

Java requires that the this(arg-list) statement appear first in the constructor before any other executable statements. (build a default object first) Use this(arg-list) as much as possible if there is multiple constructor.

Fast Encapsulation Using this Reference

- Converting a non-object-oriented programming to object-oriented programming efficiently. Direct copying the code and hook up with the instance variable using this reference.

Data Containers

LECTURE 9

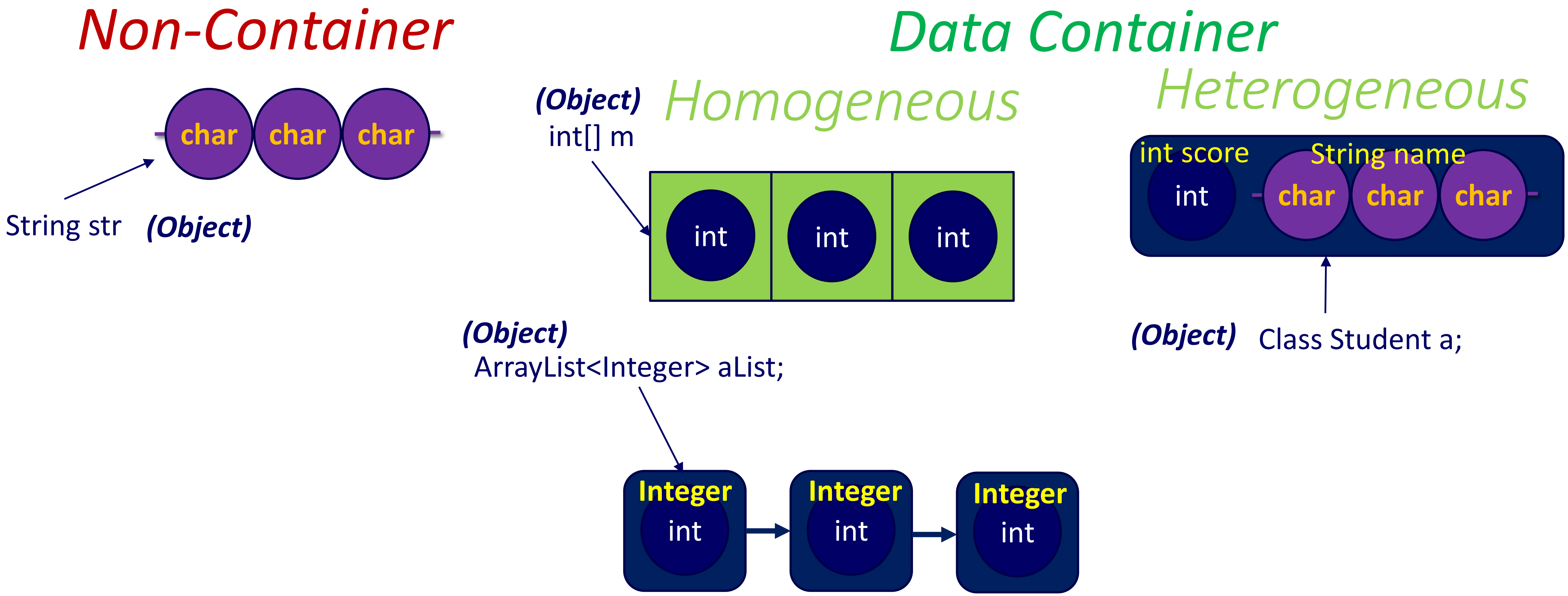
Object is a Heterogeneous Data Record

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 container

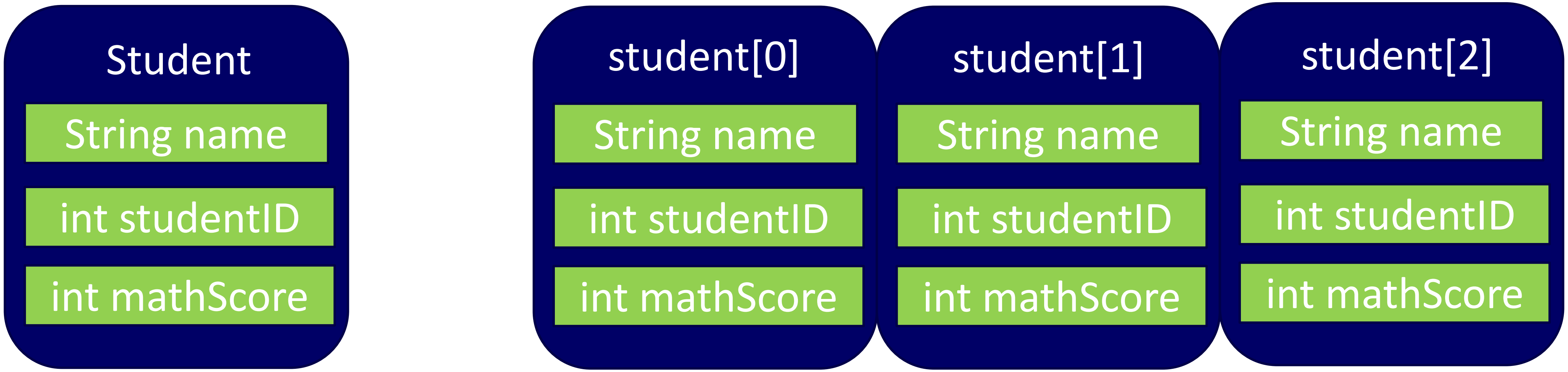
(String is immutable and can not store pointers)



Array of Objects

Student Class

Student[] students = new Student[3];



Student

String name

int studentID

int mathScore

student[0]

String name

int studentID

int mathScore

student[1]

String name

int studentID

int mathScore

student[2]

String name

int studentID

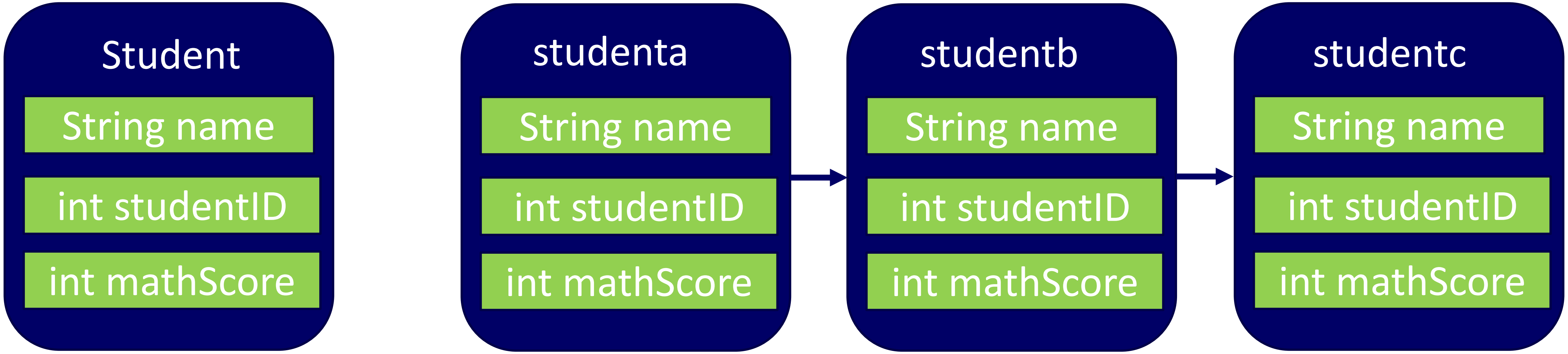
int mathScore

ArrayList of Objects

Student Class

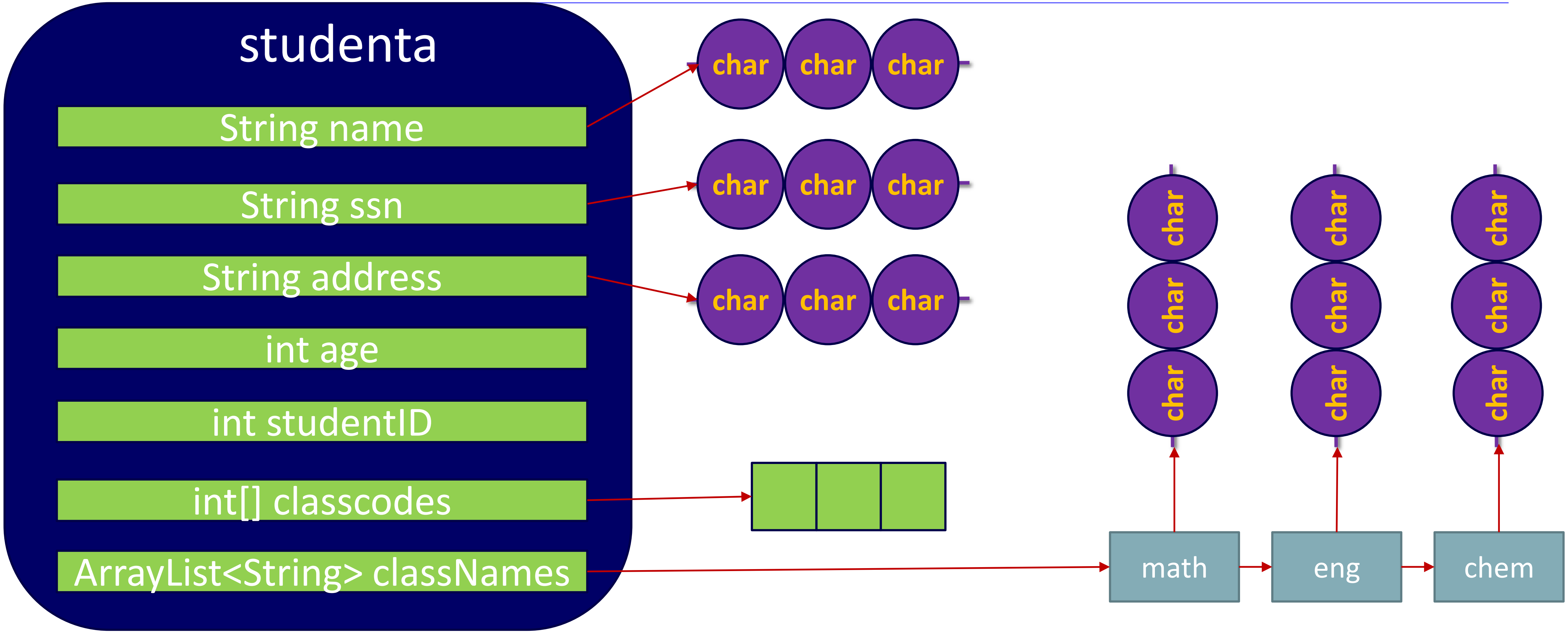
ArrayList<Student> al = new ArrayList<Student>();

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 10

Array of Objects (ArrayList of Objects)

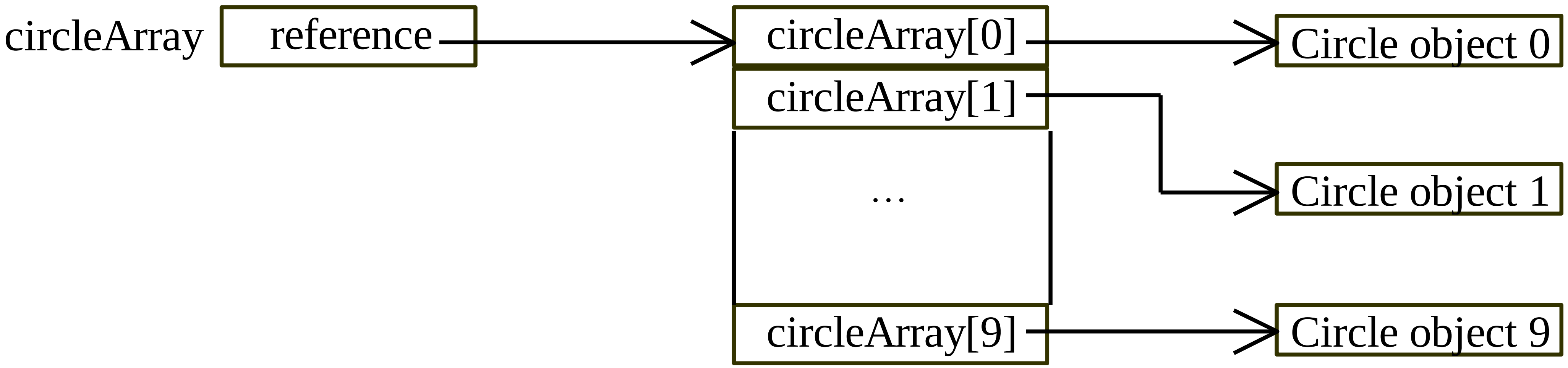
```
Circle[] circleArray = new Circle[10];
ArrayList<Circle> circleArrayList = new ArrayList<Circle>();
```

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)

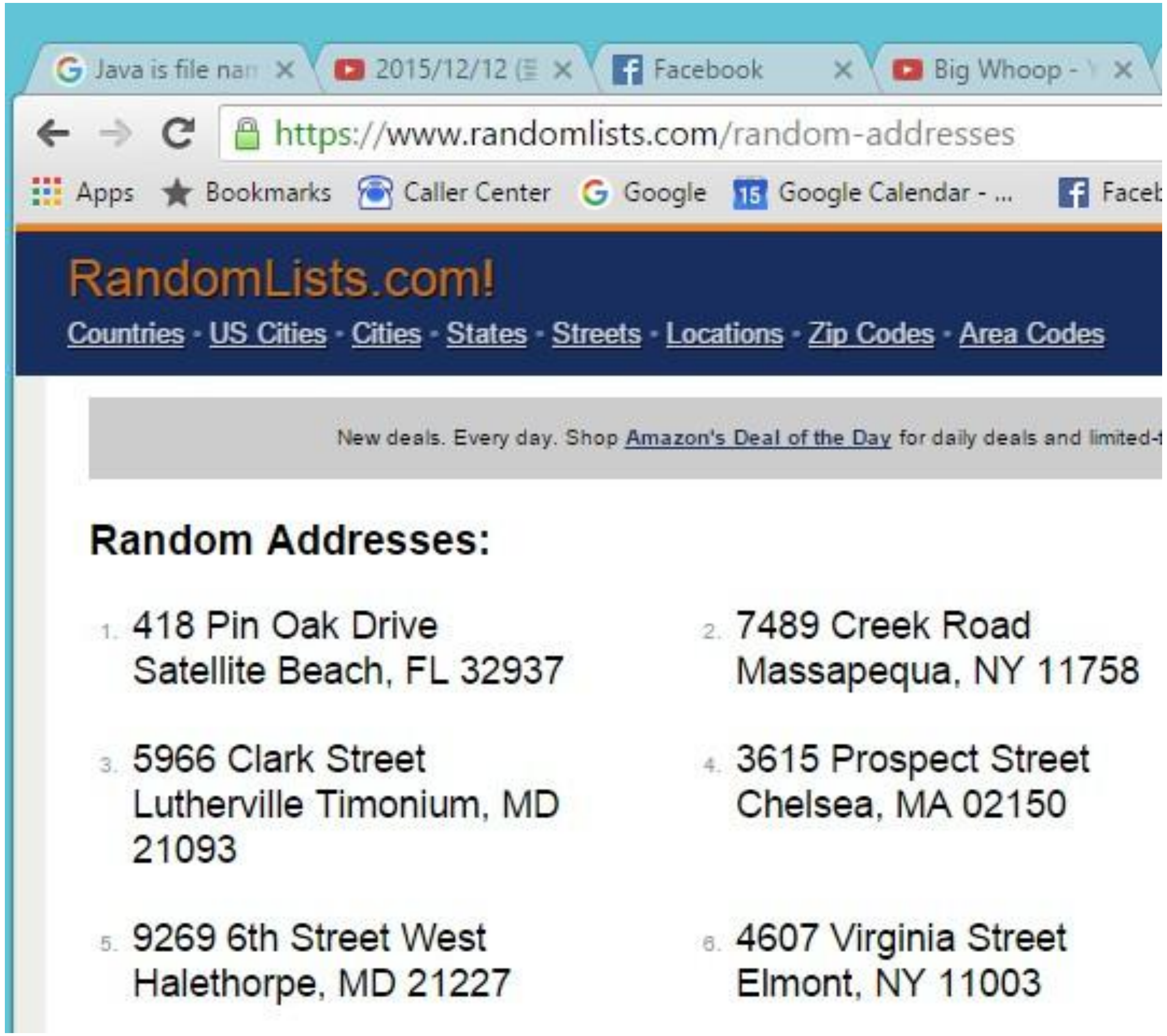
(1) Integration of StudentGPA.java (Ch. 3),
StudentInfoAnswer.java (Ch.3),
StudentGPASimulationMode.java (Ch. 4),
StudentGPAMethod.java (Ch. 6),
StudentScore.java (Ch. 7),
StudentAnswer.java (Ch. 9),
StudentScoreMultiple.java (Ch. 9)

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


Java is file nan
2015/12/12 (
Facebook
Big Whoop -
https://www.randomlists.com/random-addresses
Apps
Bookmarks
Caller Center
Google
Google Calendar - ...
Faceb
RandomLists.com!
Countries · US Cities · Cities · States · Streets · Locations · Zip Codes · Area Codes

New deals. Every day. Shop Amazon's Deal of the Day for daily deals and limited-t

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

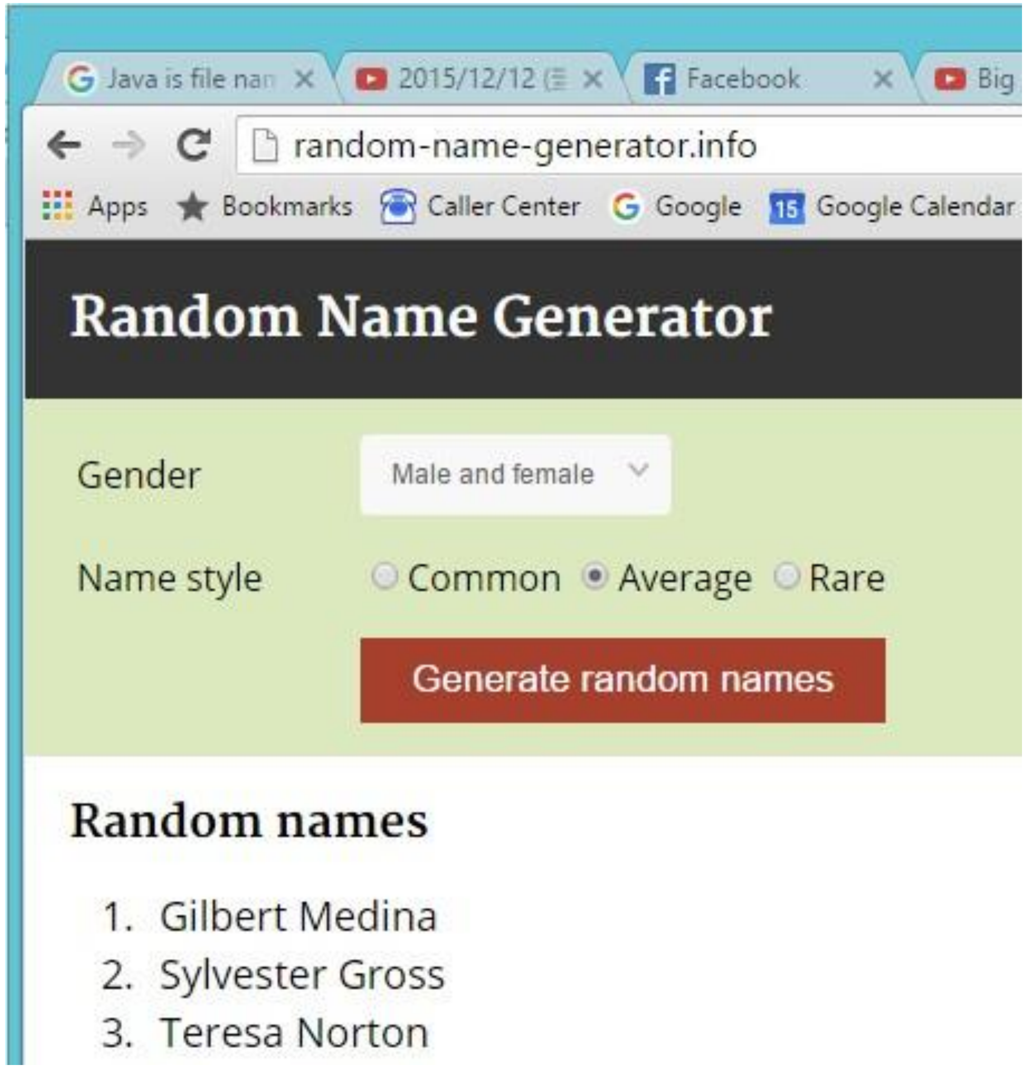
6. 4607 Virginia Street
Elmont, NY 11003

Public Domain Random Data Generators

Random Address Generator

Public Domain Random Data Generators

Random Name Generator



Random Name Generator

Gender: Male and female

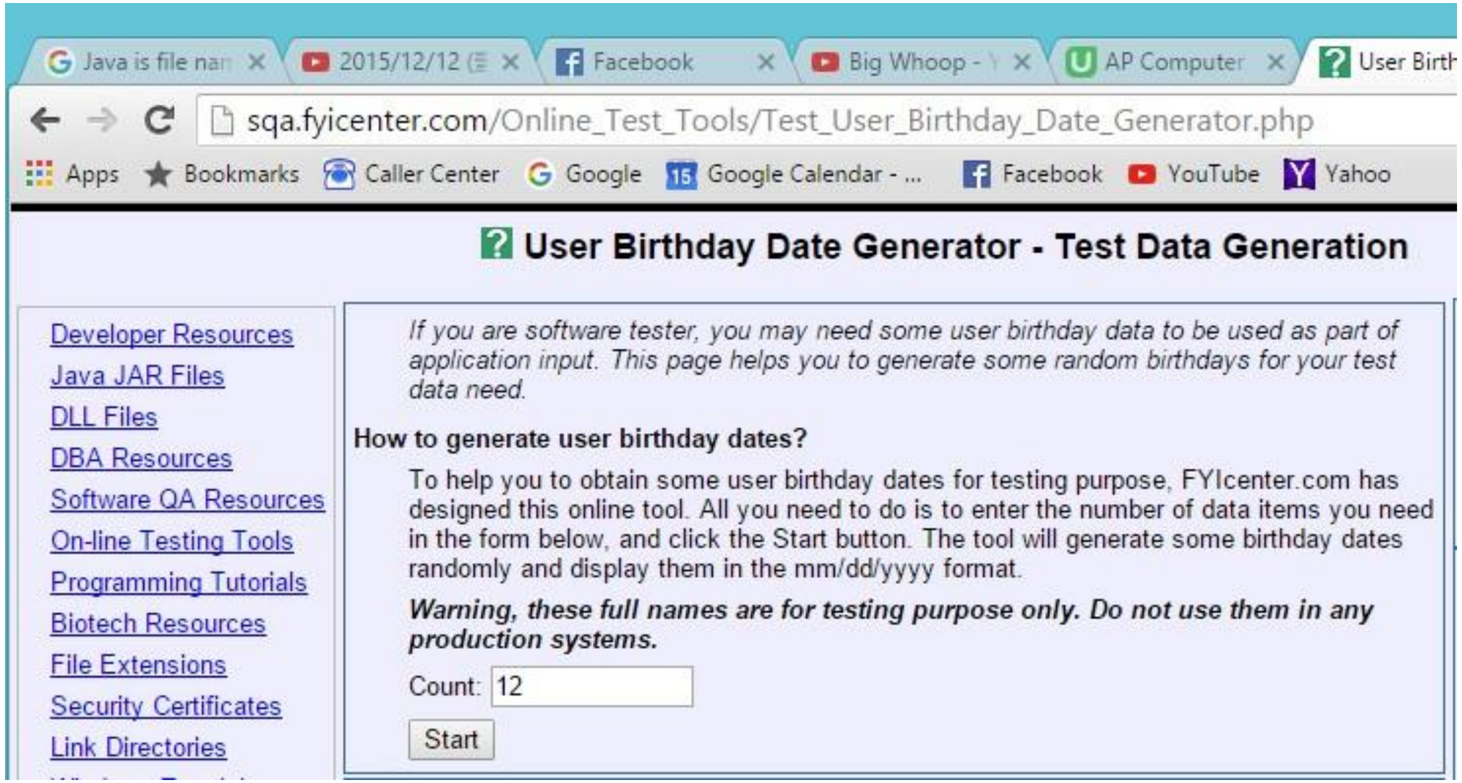
Name style: ○ Common ● Average ○ Rare

Random names

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

Public Domain Random Data Generators

Random Birthday Generator



User Birthday Date Generator - Test Data Generation

Developer Resources
Java JAR Files
DLL Files
DBA Resources
Software QA Resources
On-line Testing Tools
Programming Tutorials
Biotech Resources
File Extensions
Security Certificates
Link Directories

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.

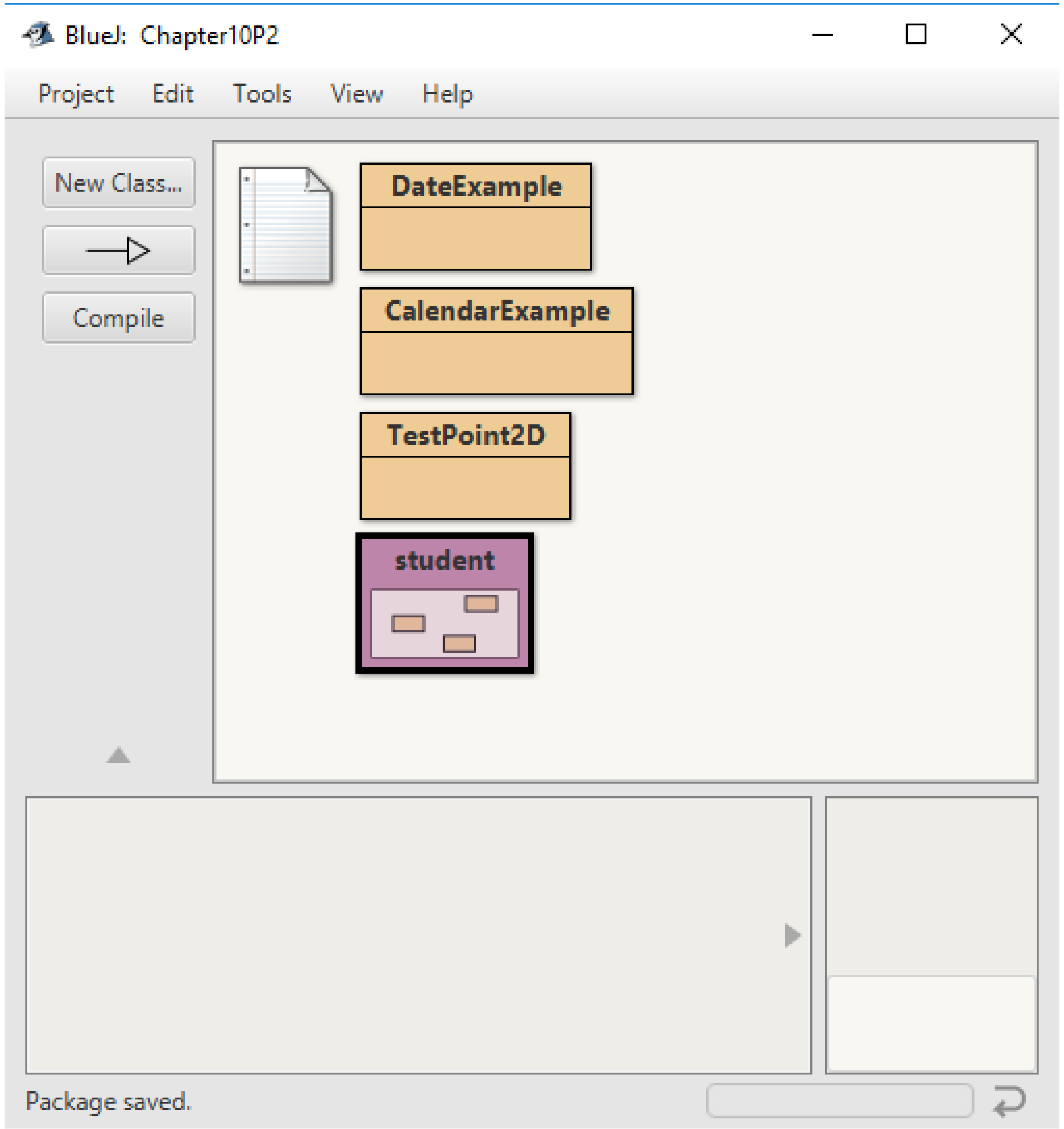
How to generate user birthday dates?

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.

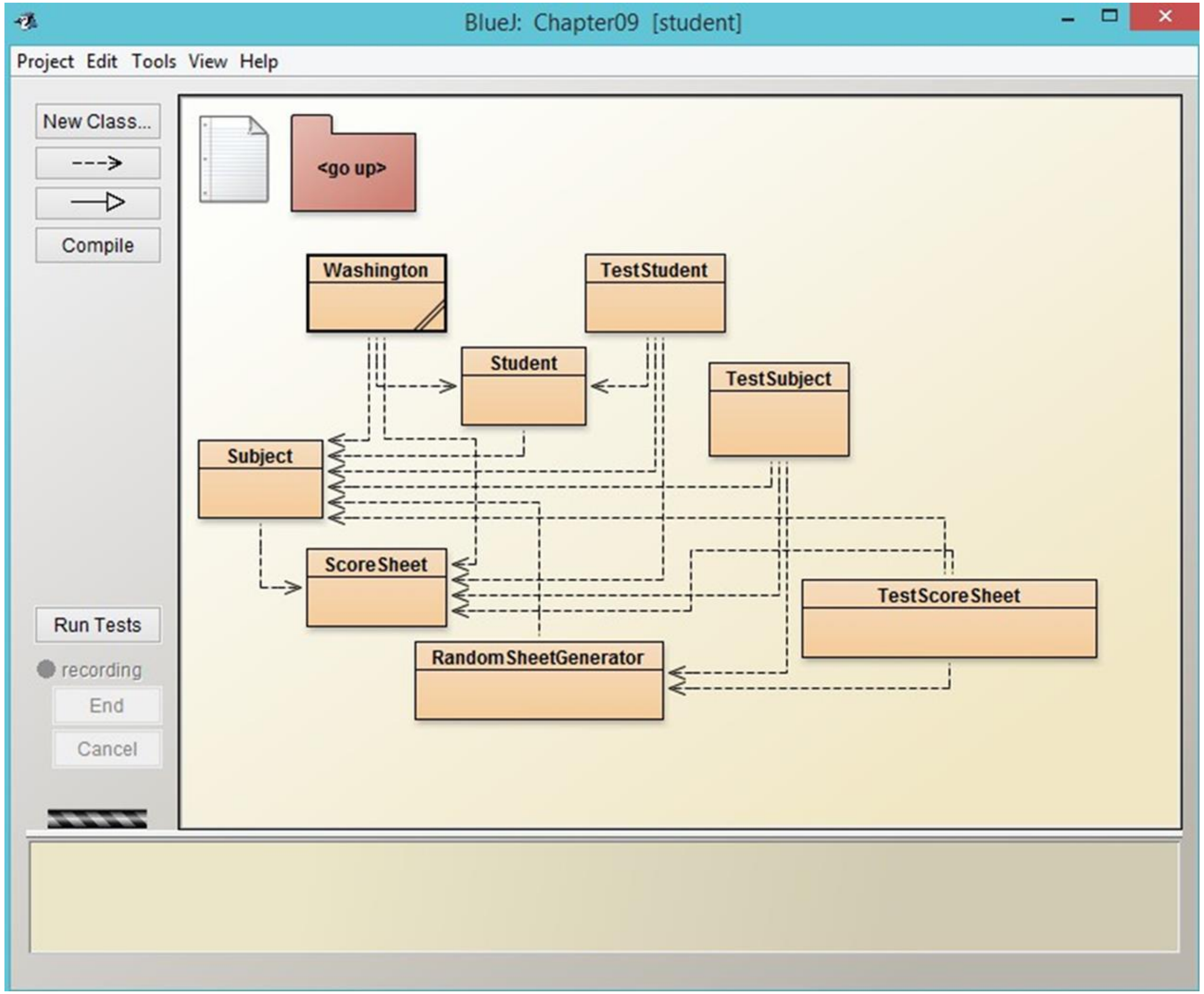
Warning, these full names are for testing purpose only. Do not use them in any production systems.

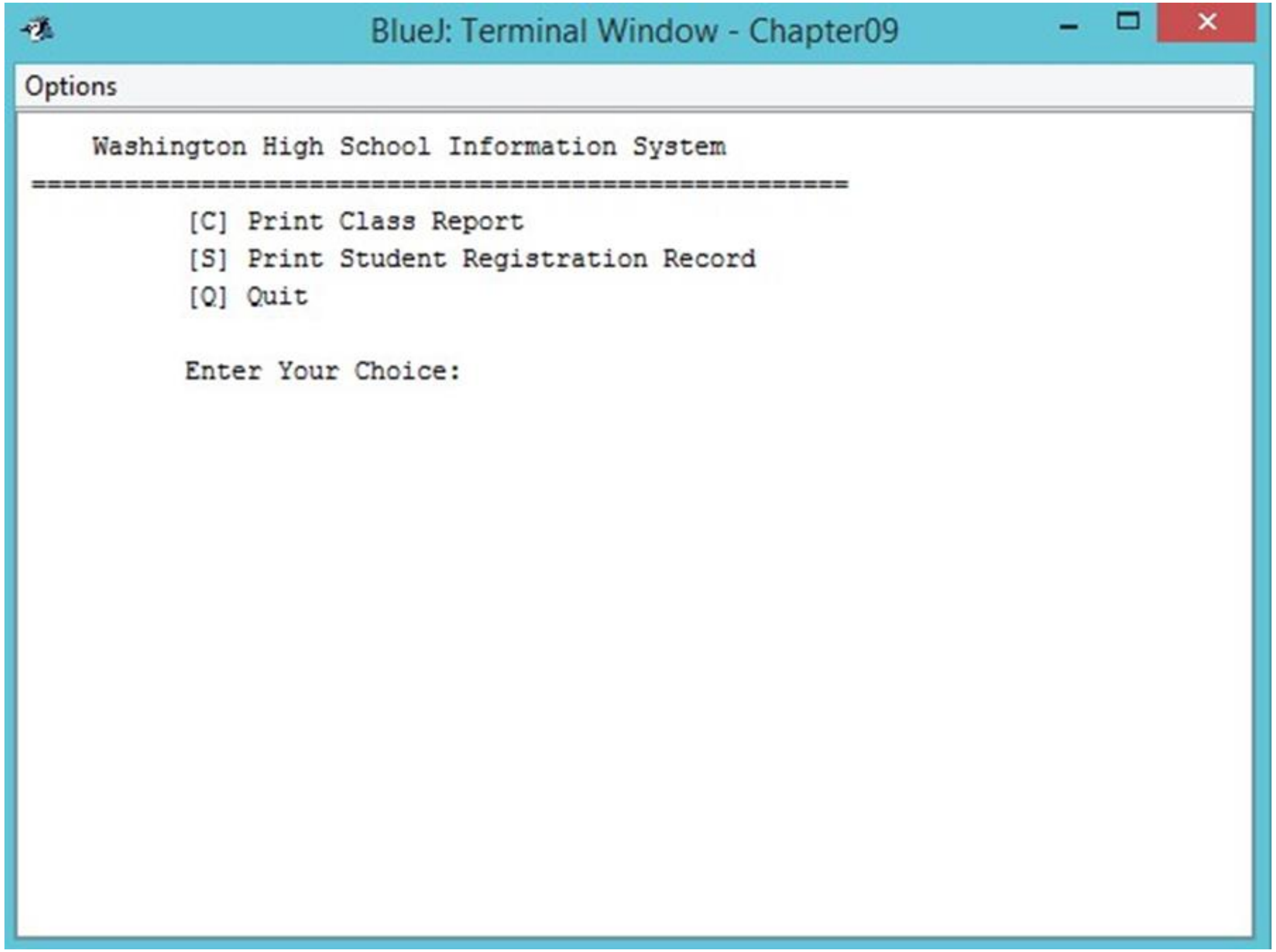
Count: 12

Start



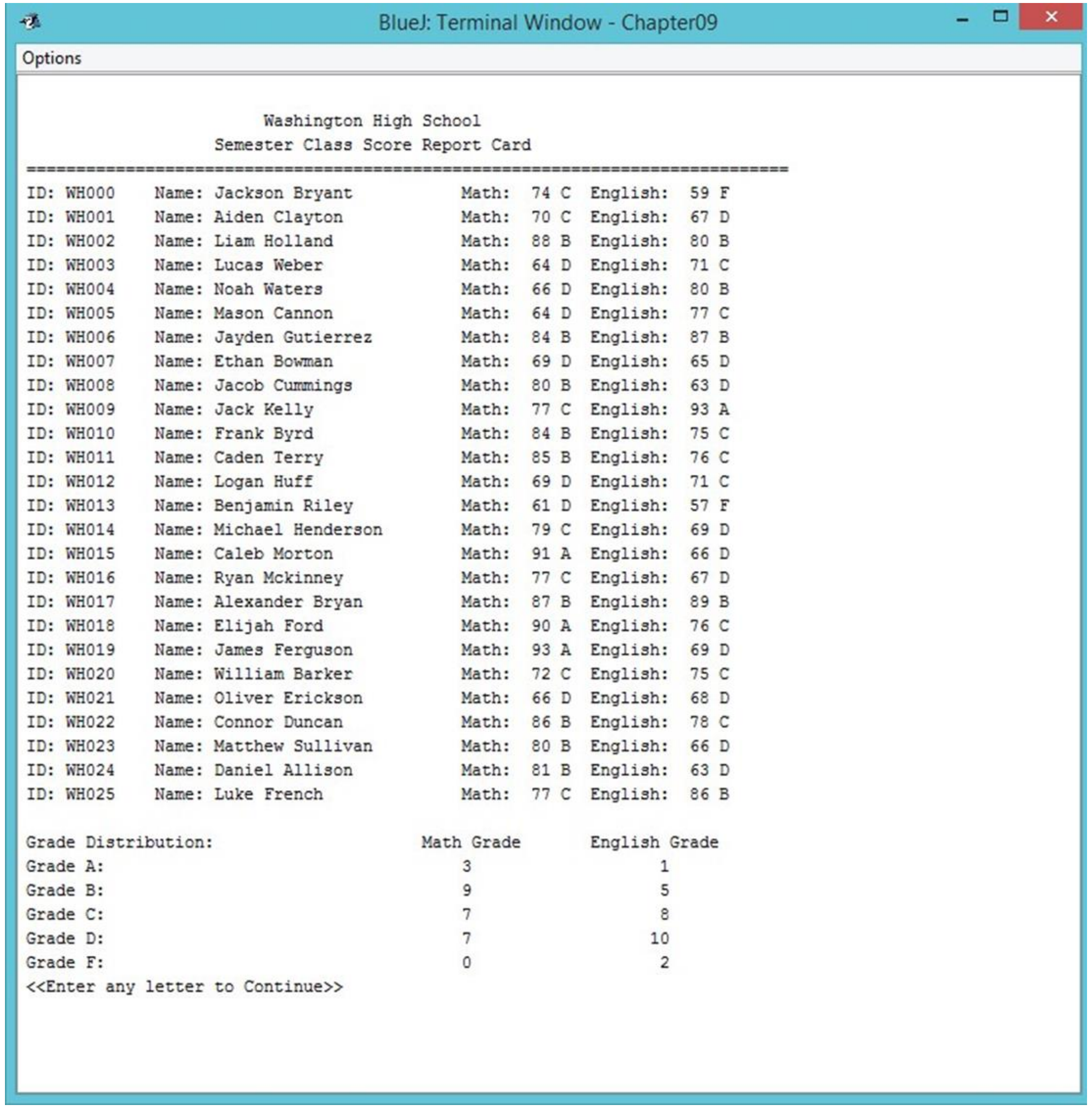
Package saved.


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

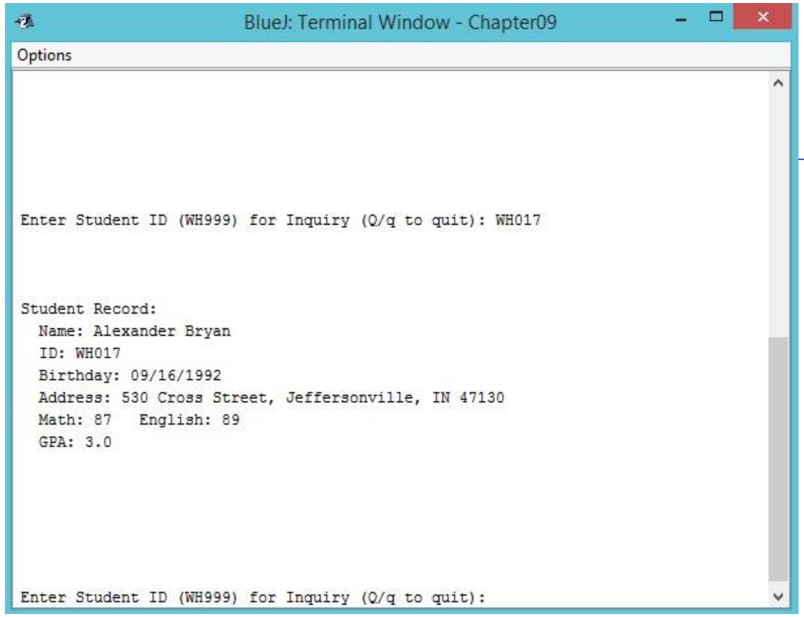


```
    Washington High School Information System
=========================================================
          [C] Print Class Report
          [S] Print Student Registration Record
          [Q] Quit

          Enter Your Choice:
```

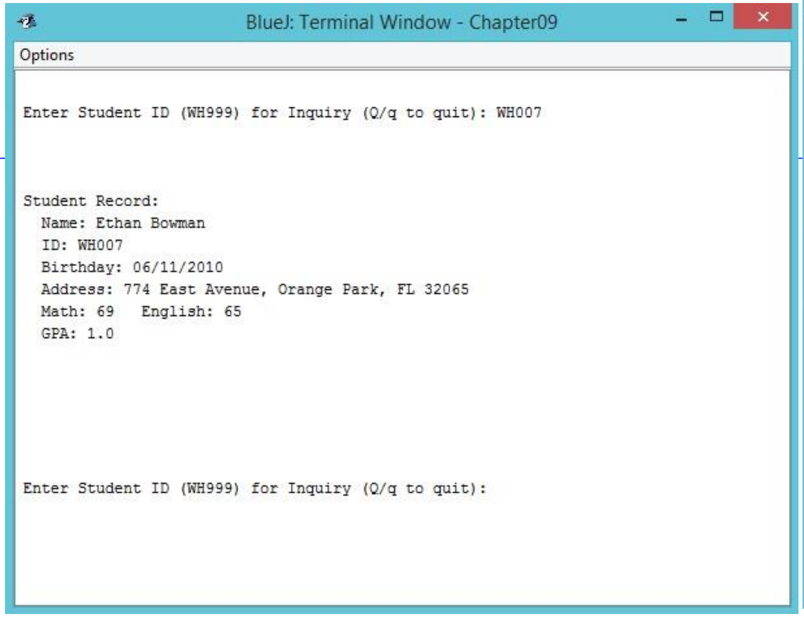
Washington High School **Welcome Manual**



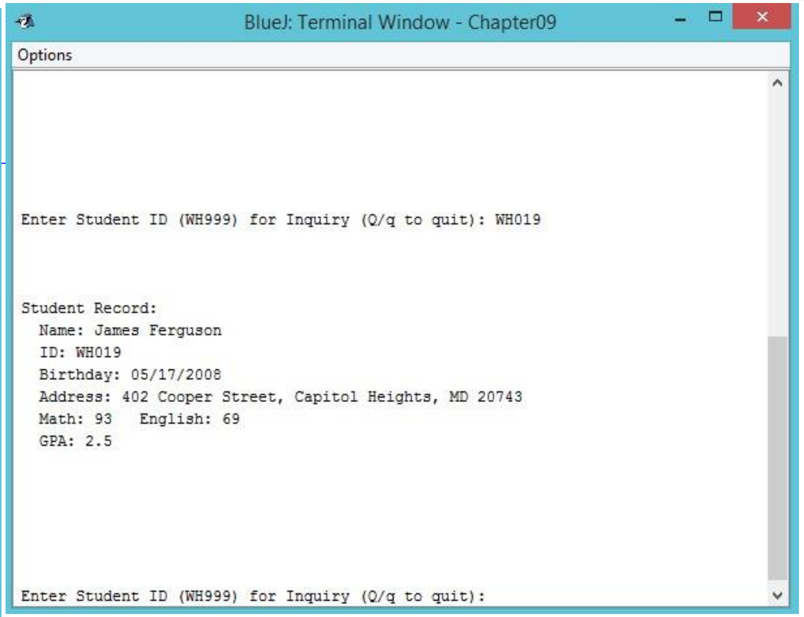
```
                        Washington High School
                   Semester Class Score Report Card
=============================================================================
ID: WH000    Name: Jackson Bryant           Math:  74 C  English:  59 F
ID: WH001    Name: Aiden Clayton            Math: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



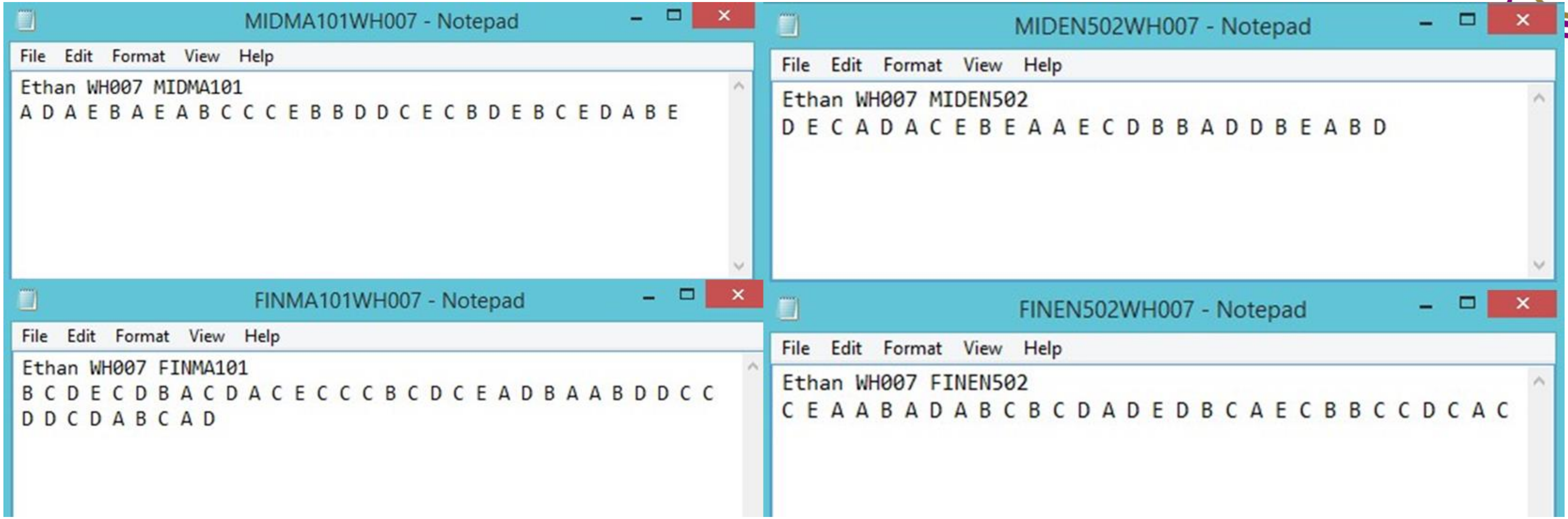
```
Enter Student ID (WH999) for Inquiry (Q/q to quit): WH017


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: Class using Array or ArrayList

LECTURE 11

Baseball Team

- Baseball team is a class that use an Array of Players

```
class Baseball{
   Player[] plist;

   Baseball(){ plist = new Player[9]; }
   Baseball(Plyaer[] pp){
       plist = pp;
   }
}
```

Basketball Team

- Basketball team is a class that use an Arraylist of Players

```
class Basketball{
   ArrayList<Player> plist;

   Baseball(){ plist = new ArrayList<Player>(); }
   Baseball(ArrayList<Player> pp){
       plist = pp;
   }
}
```

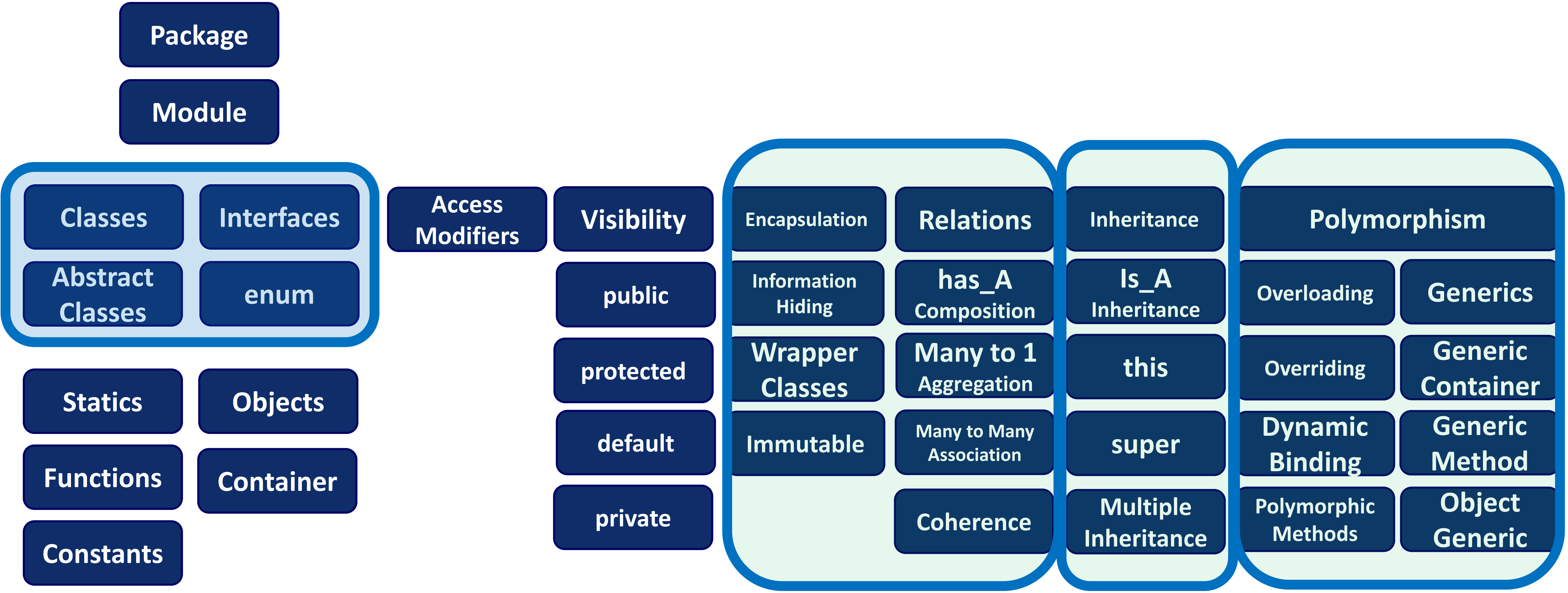
Summary

LECTURE 12

Summary

- Class and object design provide many possibility to enhance data abstraction.
- The purpose of data abstraction is to make the data more reusable, modular, maintainable, readable.
- Data Encapsulation, Immutability, Arrays of Object, Object using Arrays are many different ways of enhancing data abstraction.

Object-Oriented Programming



Package

Module

Classes

Interfaces

Abstract Classes

enum

Statics

Objects

Functions

Container

Constants

Access Modifiers

Visibility

public

protected

default

private

Encapsulation

Information Hiding

Wrapper Classes

Immutable

Relations

has_A Composition

Many to 1 Aggregation

Many to Many Association

Coherence

Inheritance

Is_A Inheritance

this

super

Multiple Inheritance

Polymorphism

Overloading

Generics

Overriding

Generic Container

Dynamic Binding

Generic Method

Polymorphic Methods

Object Generic