AP Computer Science B

Java Object-Oriented Programming [Ver. 2.0]

Unit 4: Object-Oriented Design

WEEK 1: CHAPTER 10 CLASSES AND OBJECTS (PART 1 PROGRAM AND DATA)

DR. ERIC CHOU IEEE SENIOR MEMBER

Objectives

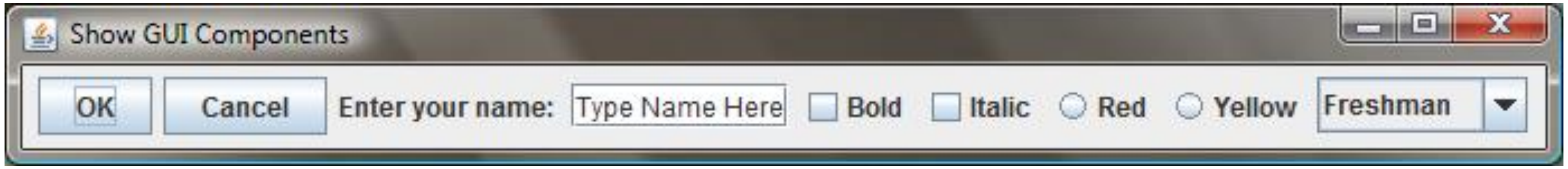
- Class Definition – Data Fields, Constructor, and Member Methods
- Object Declaration – Instantiation and Initialization
- Static Programming
- Object-Oriented Programming: Class, Package
- Class, Reference, and Objects

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

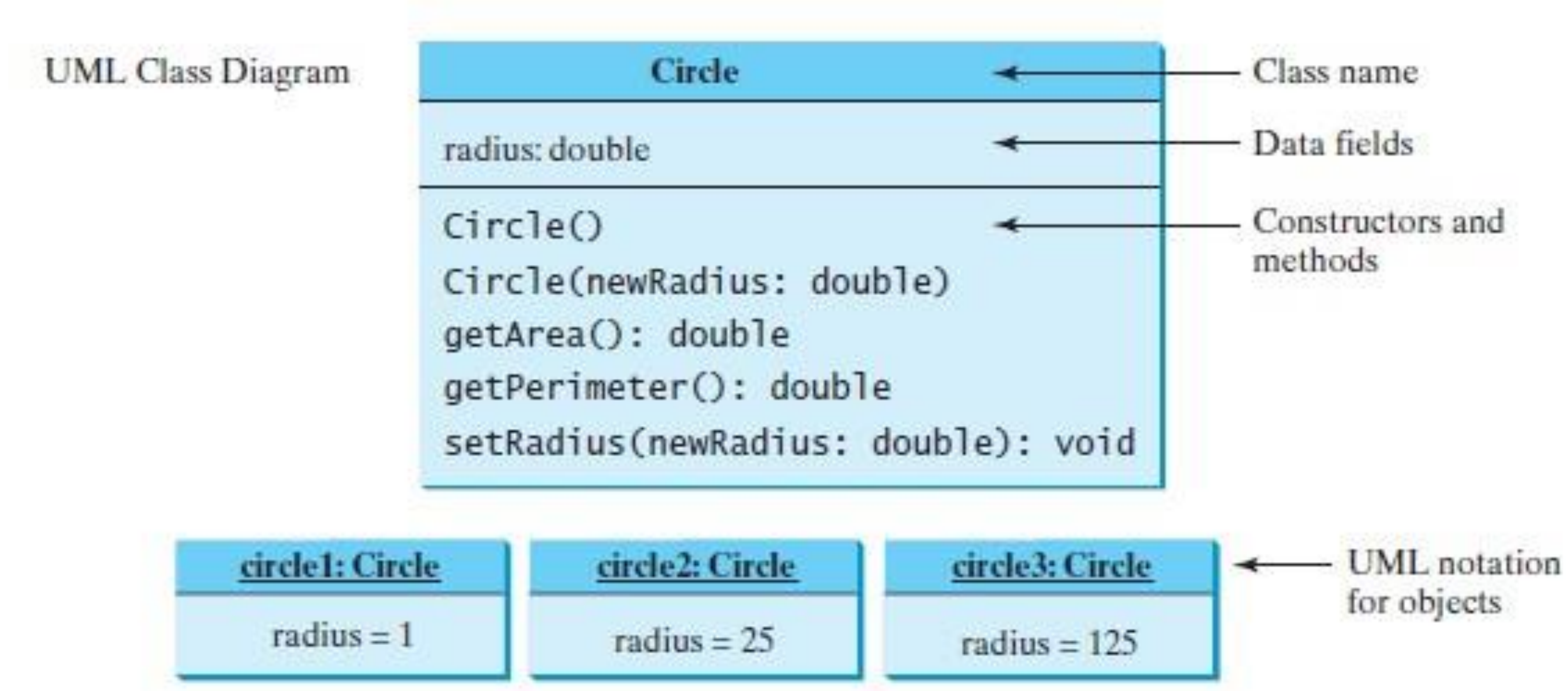
OO Programming Concepts

- 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.

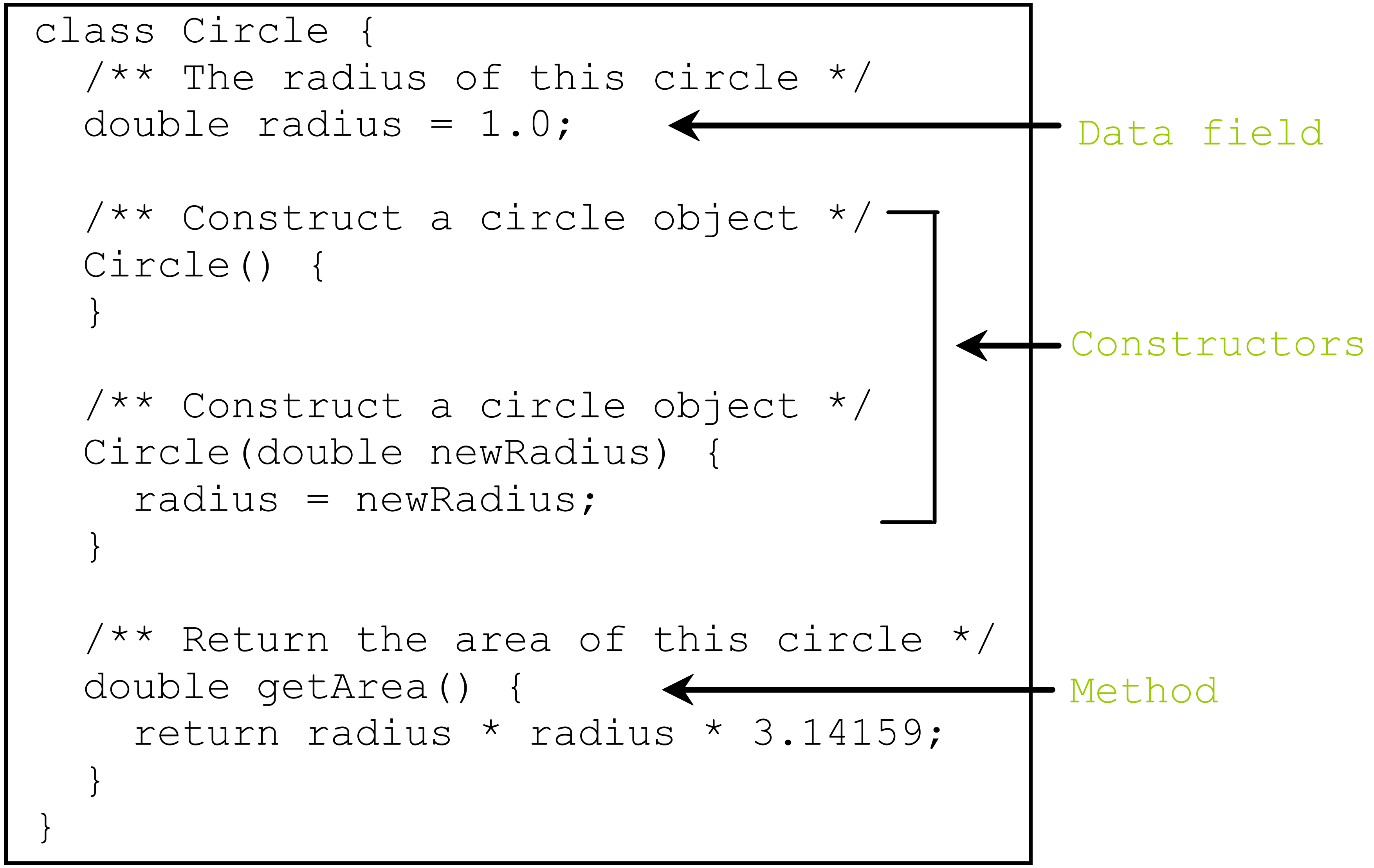
Objects: UML Class/Object Diagram



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`

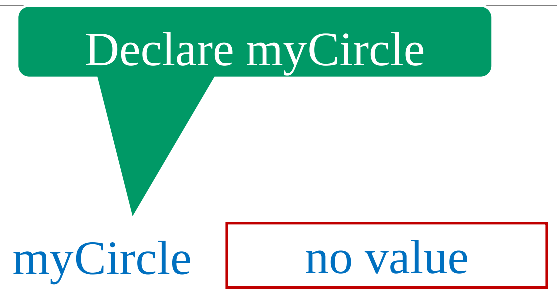
Invoking the object's method:

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

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

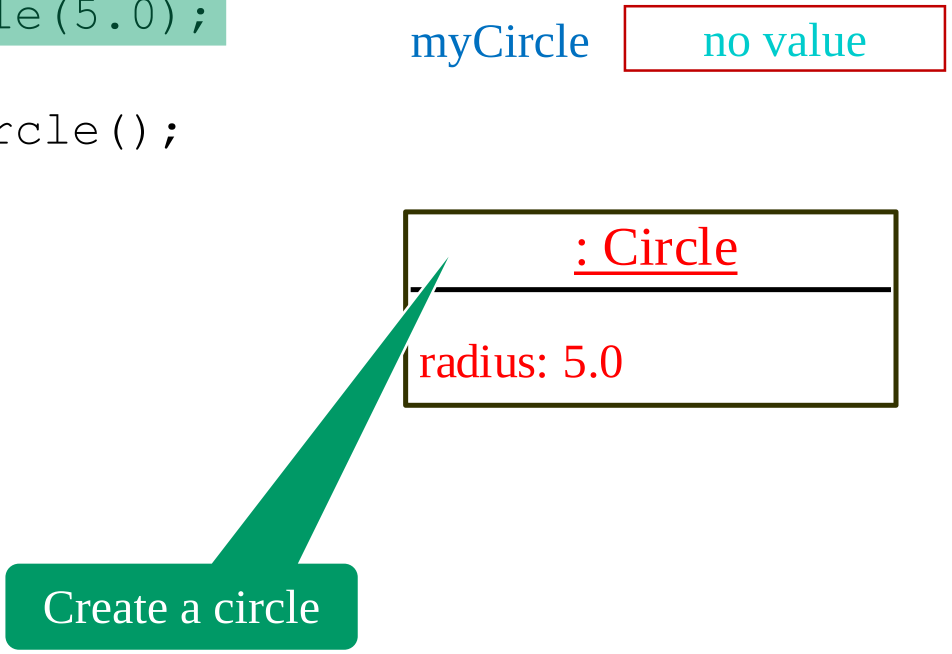
Trace Code

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



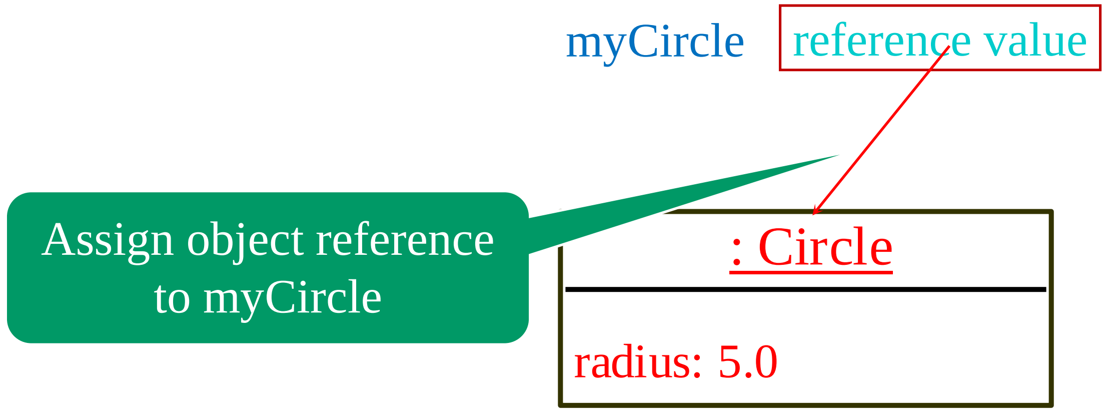
Trace Code

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



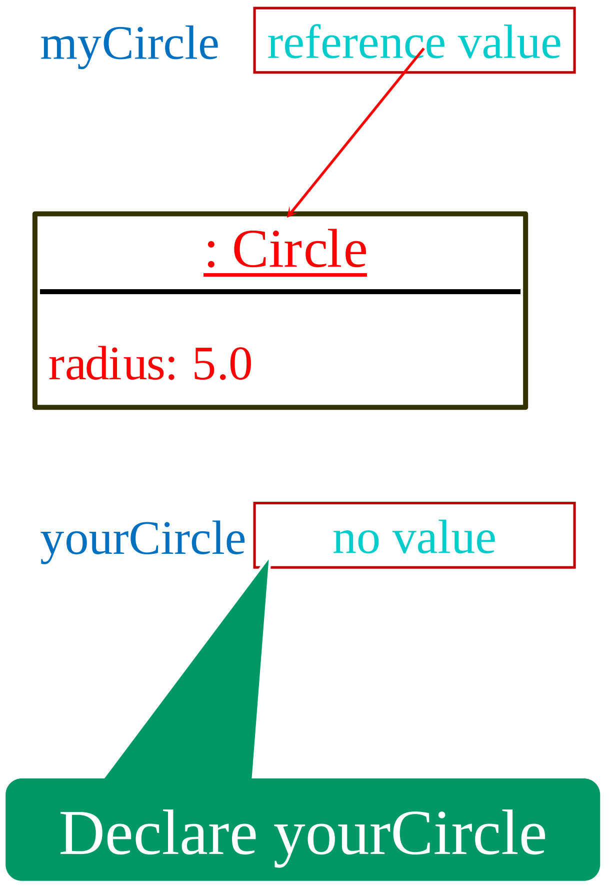
Trace Code

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



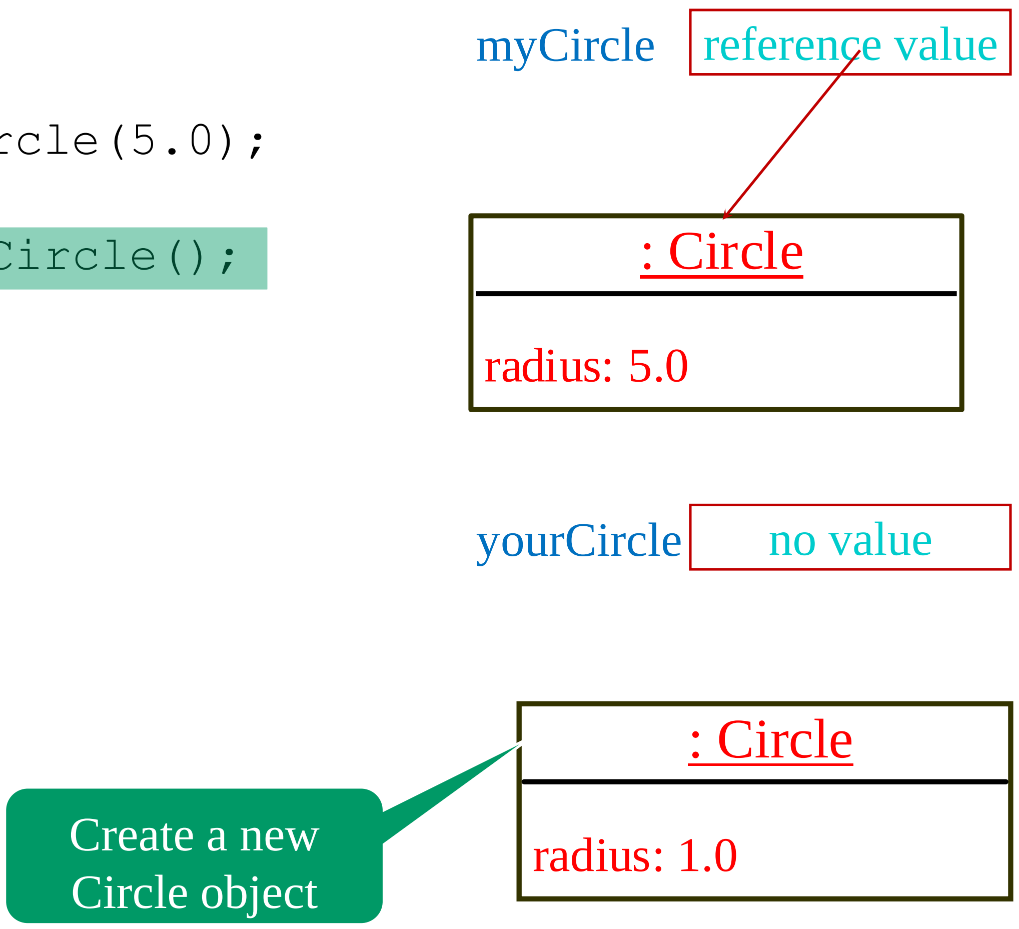
Trace Code

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



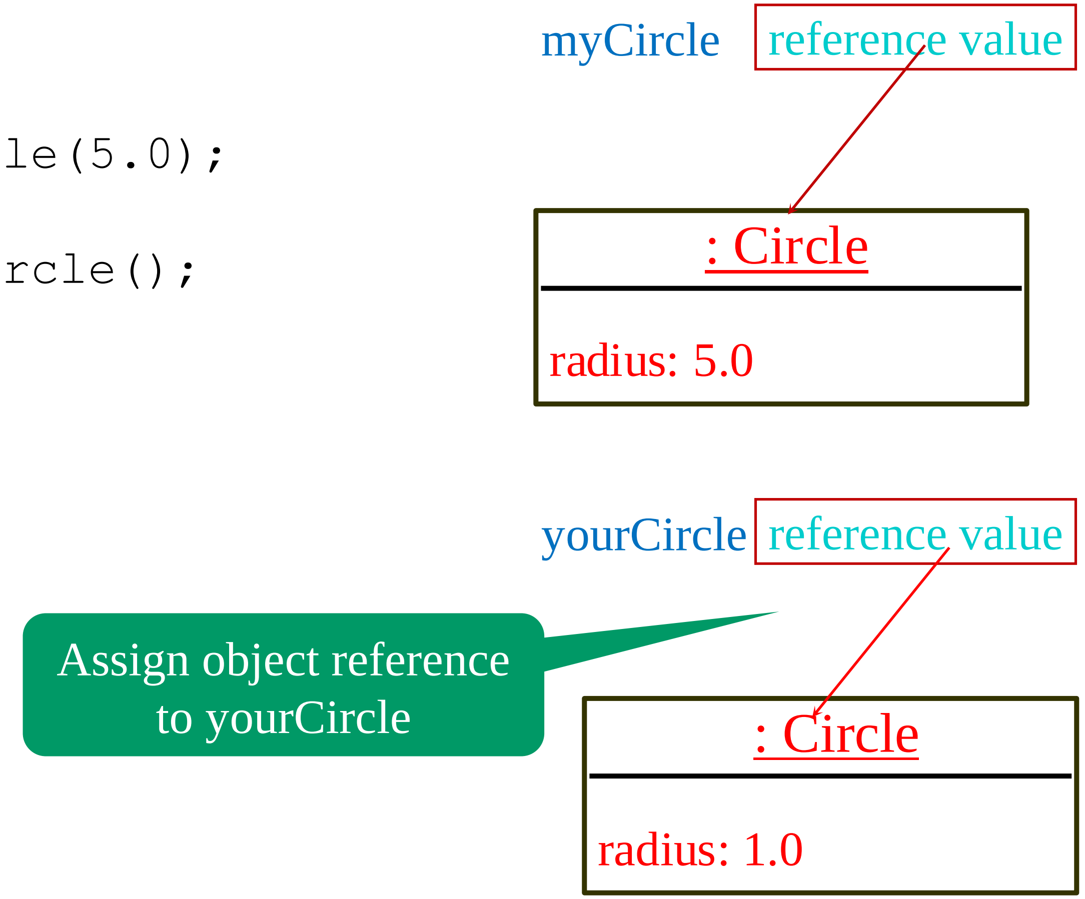
Trace Code

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



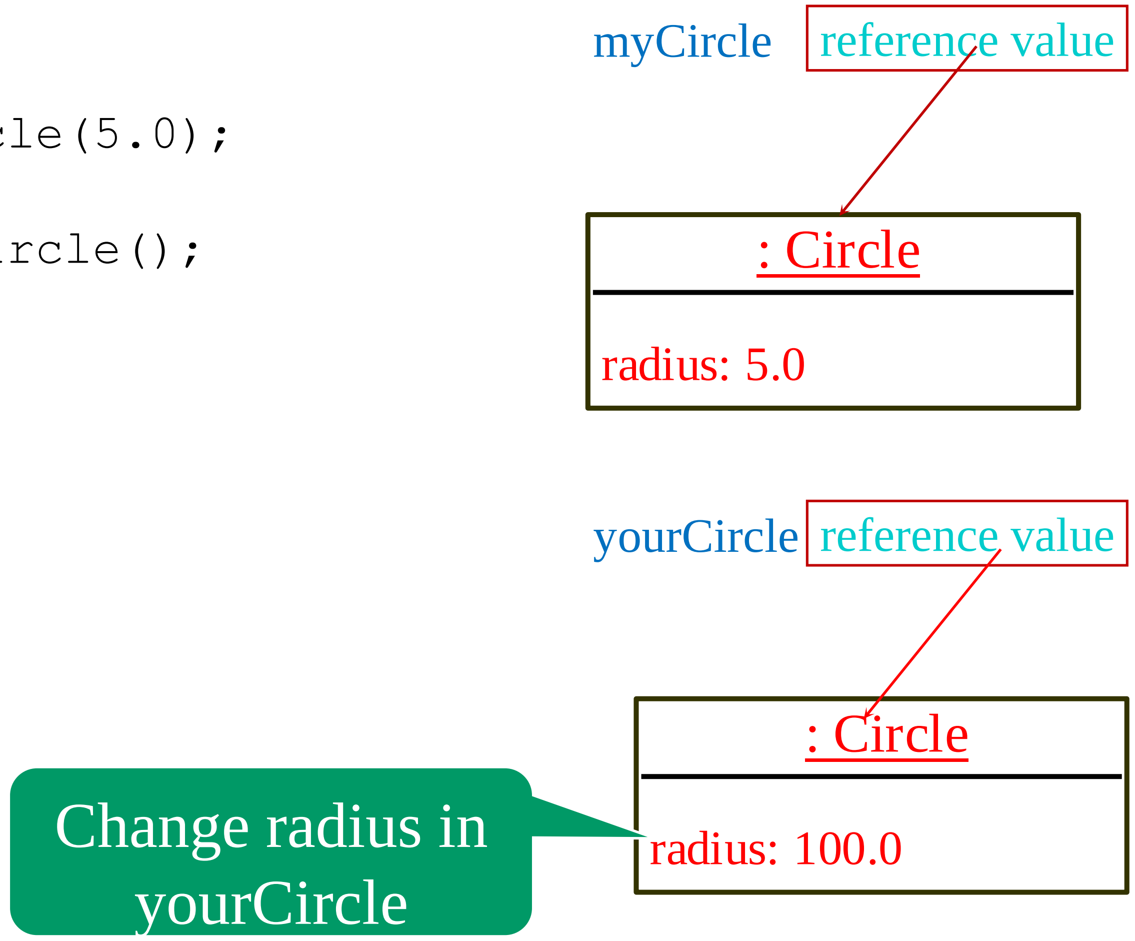
Trace Code

```
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;
```



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."

Demo Program:

Creation of a Package Complex and Constructors

LECTURE 3

Package (.jar library file)

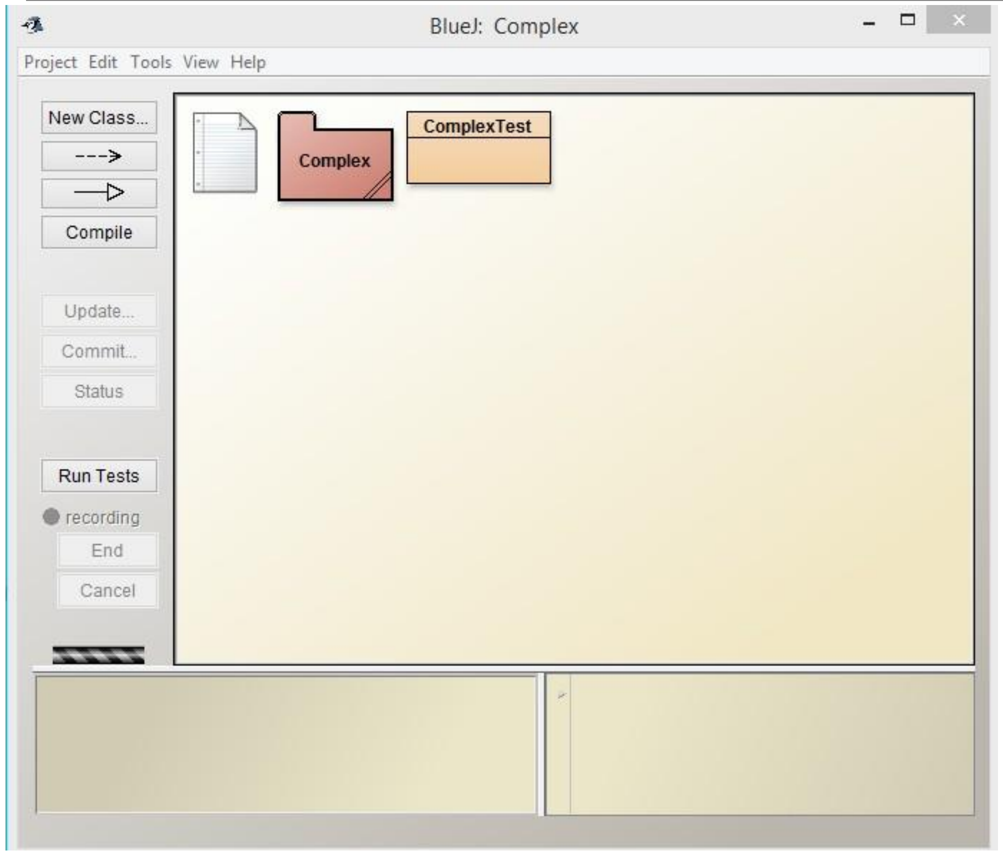
- A **package** is a collection of programs or program modules (classes).
- **BlueJ** use a 3-level hierarchy:
- **project --- package --- class**

- **Package** management varies from IDE to IDE. So, you have to refer to each IDE tool user's manual.
- In this presentation, we will be focused on **BlueJ**.

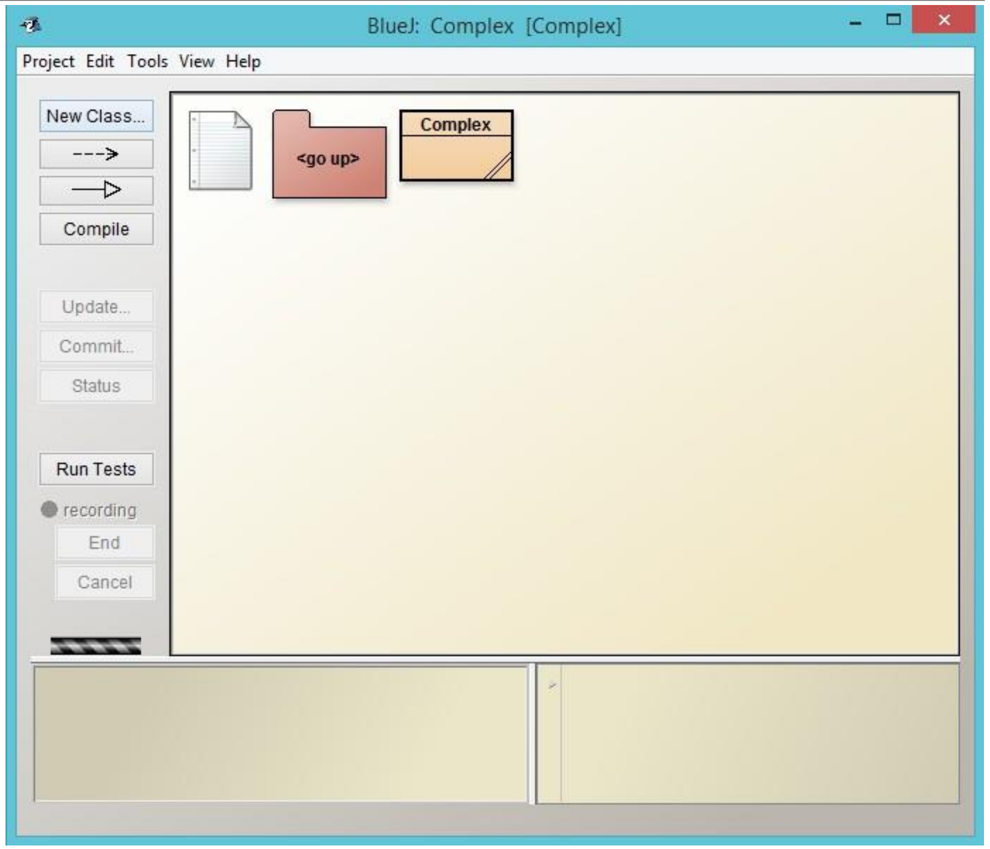
e.g.

java is a project, **java.util** is a package, while **java.util.Scanner** is a class.

Package and Class

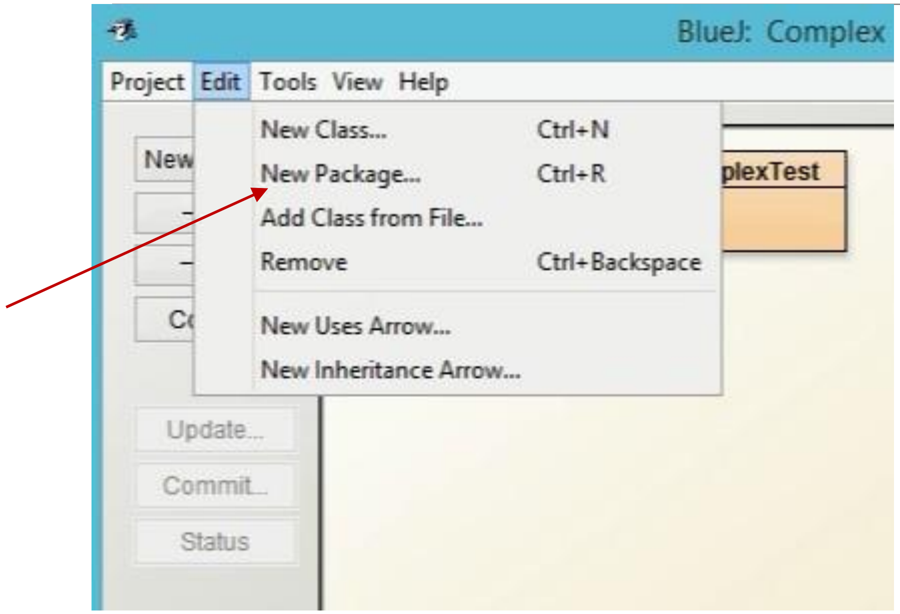


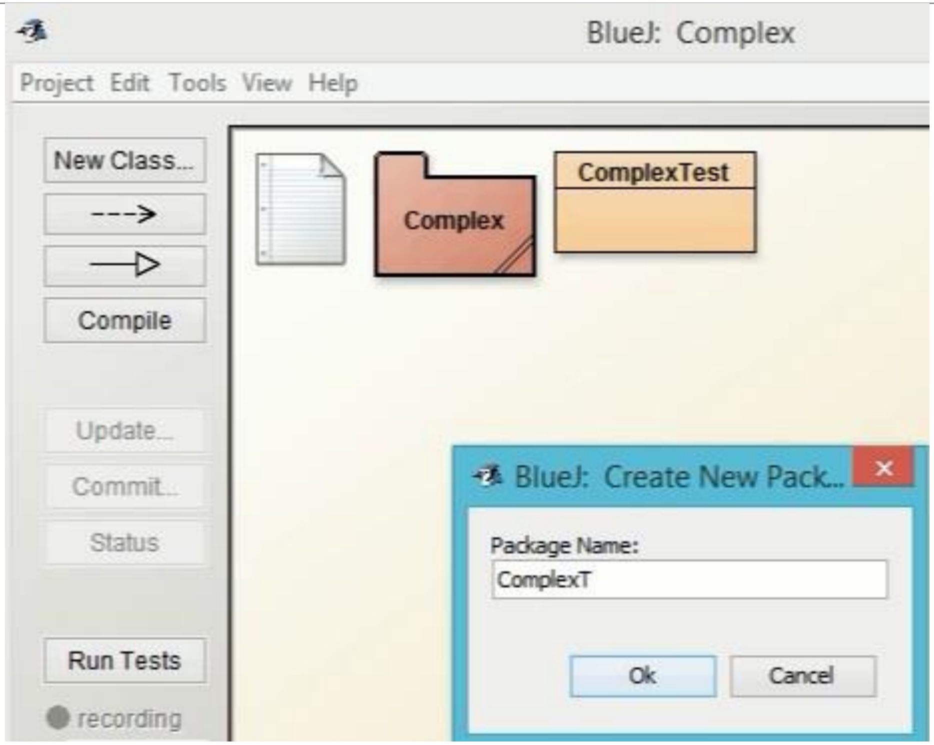
Project View



Package View

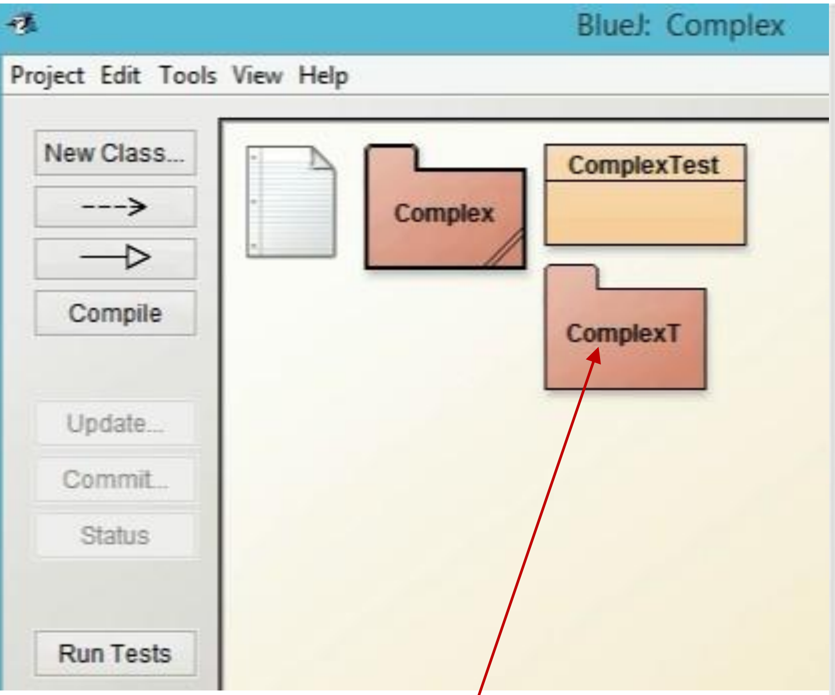
Creation of Package



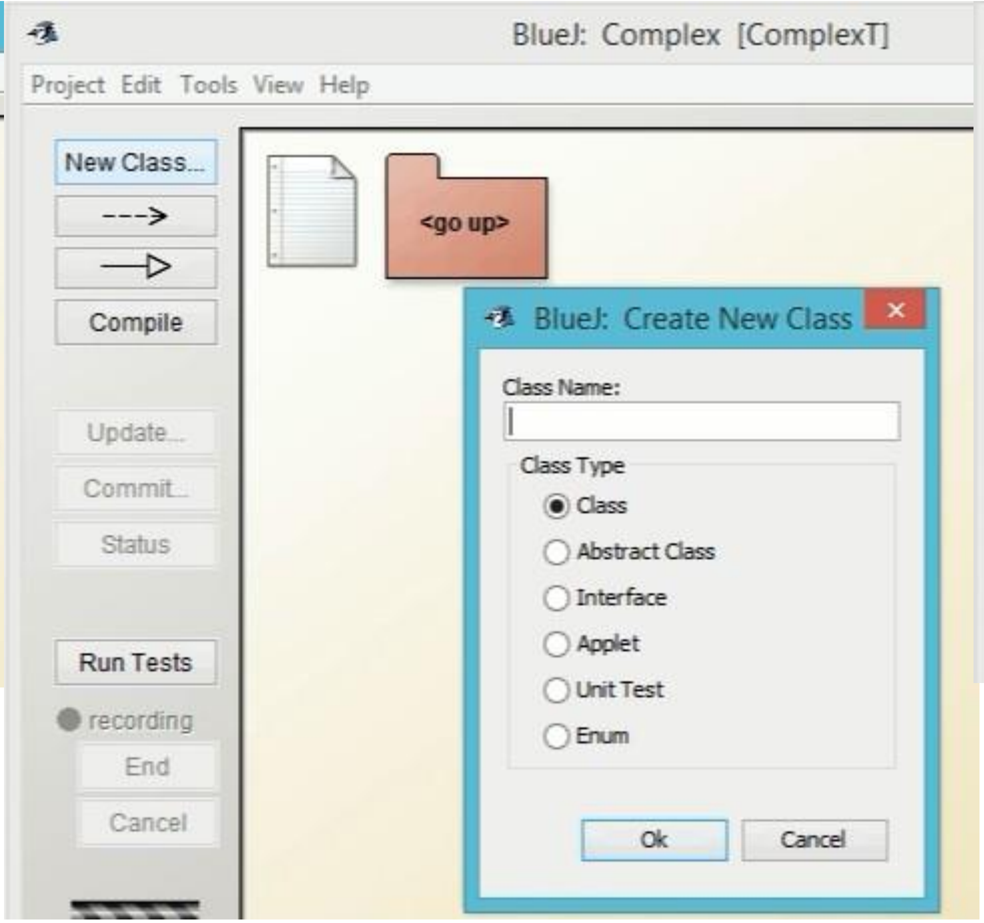


Package Name: ComplexT

Create class in a package



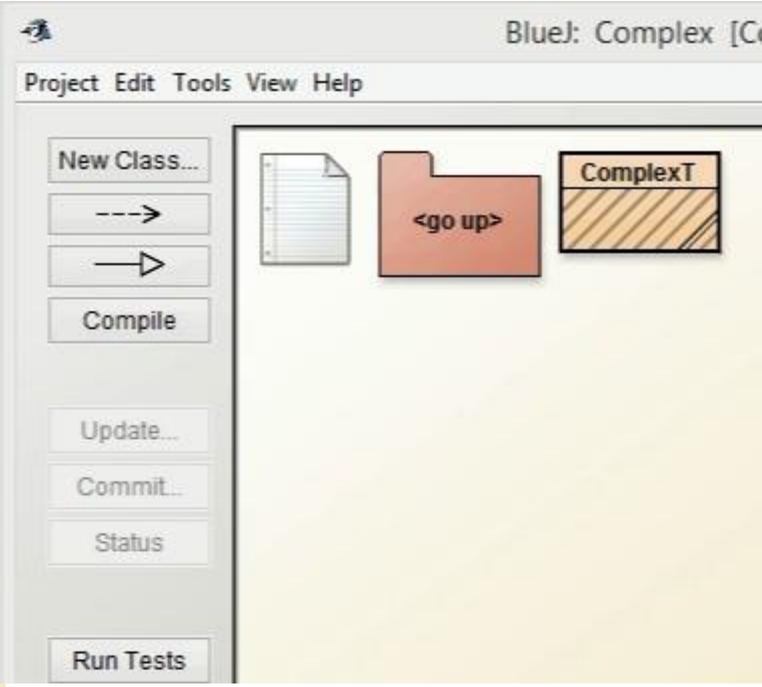
Click Here to enter package view



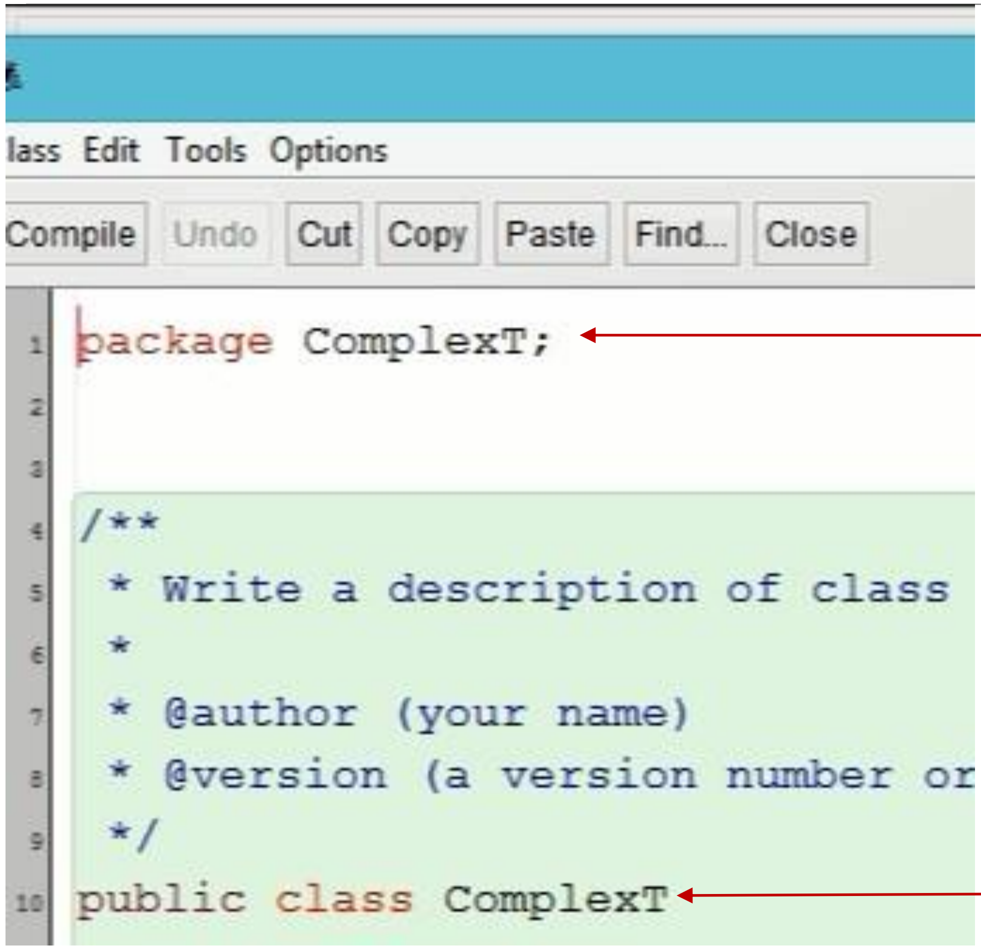
Class Name:

Class Type

- [✓ Class]
- Abstract Class
- Interface
- Applet
- Unit Test
- Enum



Package Declaration



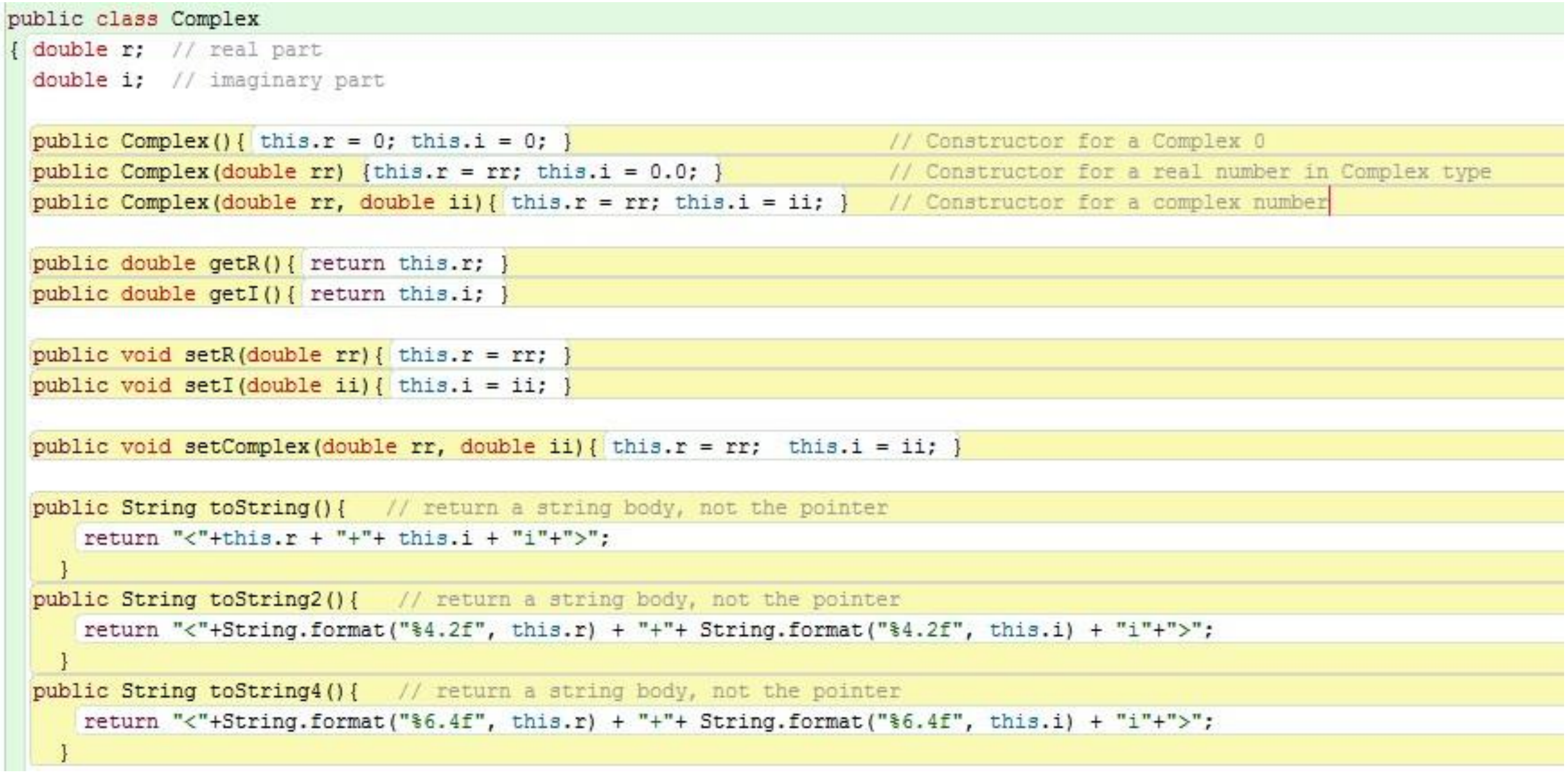
```
package ComplexT;


/**
 * Write a description of class
 *
 * @author (your name)
 * @version (a version number or
 */
public class ComplexT
```

package

class

Basic Complex Class

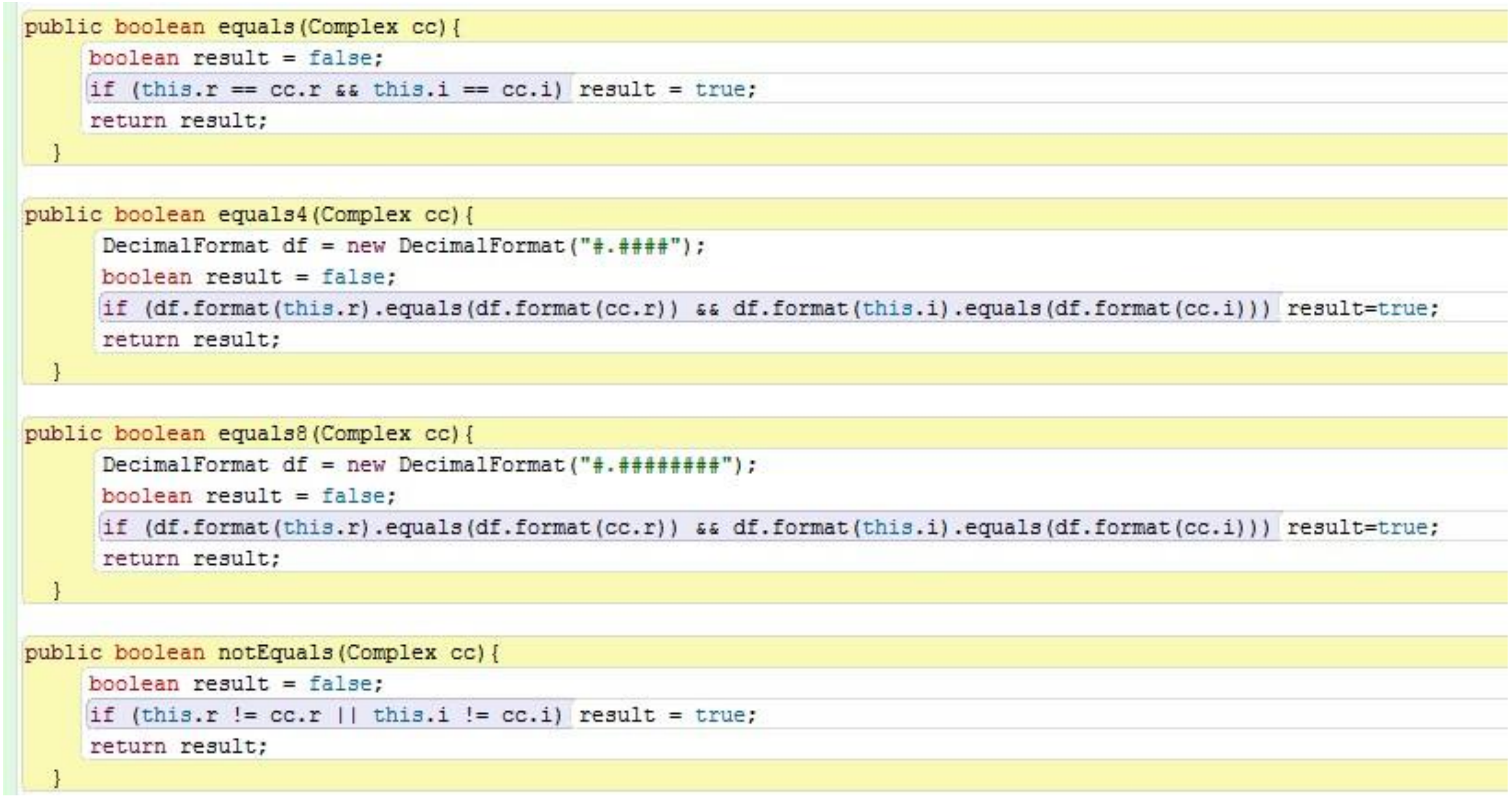
Complex.java and ComplexTest.java



```
public class Complex
{ double r;  // real part
  double i;  // imaginary part

  public Complex(){ this.r = 0; this.i = 0; }                         // Constructor for a Complex 0
  public Complex(double rr) {this.r = rr; this.i = 0.0; }             // Constructor for a real number in Complex type
  public Complex(double rr, double ii){ this.r = rr; this.i = ii; }   // Constructor for a complex number

  public double getR(){ return this.r; }
  public double getI(){ return this.i; }

  public void setR(double rr){ this.r = rr; }
  public void setI(double ii){ this.i = ii; }

  public void setComplex(double rr, double ii){ this.r = rr;  this.i = ii; }

  public String toString(){    // return a string body, not the pointer
      return "<"+this.r + "+"+ this.i + "i"+">";
   }
  public String toString2(){    // return a string body, not the pointer
      return "<"+String.format("%4.2f", this.r) + "+"+ String.format("%4.2f", this.i) + "i"+">";
   }
  public String toString4(){    // return a string body, not the pointer
      return "<"+String.format("%6.4f", this.r) + "+"+ String.format("%6.4f", this.i) + "i"+">";
   }
```

Basic Complex Class

Complex.java and ComplexTest.java



```java
public boolean equals(Complex cc){
	boolean result = false;
	if (this.r == cc.r && this.i == cc.i) result = true;
	return result;
}

public boolean equals4(Complex cc){
	DecimalFormat df = new DecimalFormat("#.####");
	boolean result = false;
	if (df.format(this.r).equals(df.format(cc.r)) && df.format(this.i).equals(df.format(cc.i))) result=true;
	return result;
}

public boolean equals8(Complex cc){
	DecimalFormat df = new DecimalFormat("#.########");
	boolean result = false;
	if (df.format(this.r).equals(df.format(cc.r)) && df.format(this.i).equals(df.format(cc.i))) result=true;
	return result;
}

public boolean notEquals(Complex cc){
	boolean result = false;
	if (this.r != cc.r || this.i != cc.i) result = true;
	return result;
}
```

Basic Complex Class

Complex.java and ComplexTest.java

// Conversion to String.

String.Format(“%8.2f”, a);

// conversion to various of formats

import java.text.DecimalFormat;

DecimalFormat df = new DecimalFormat("#.####");

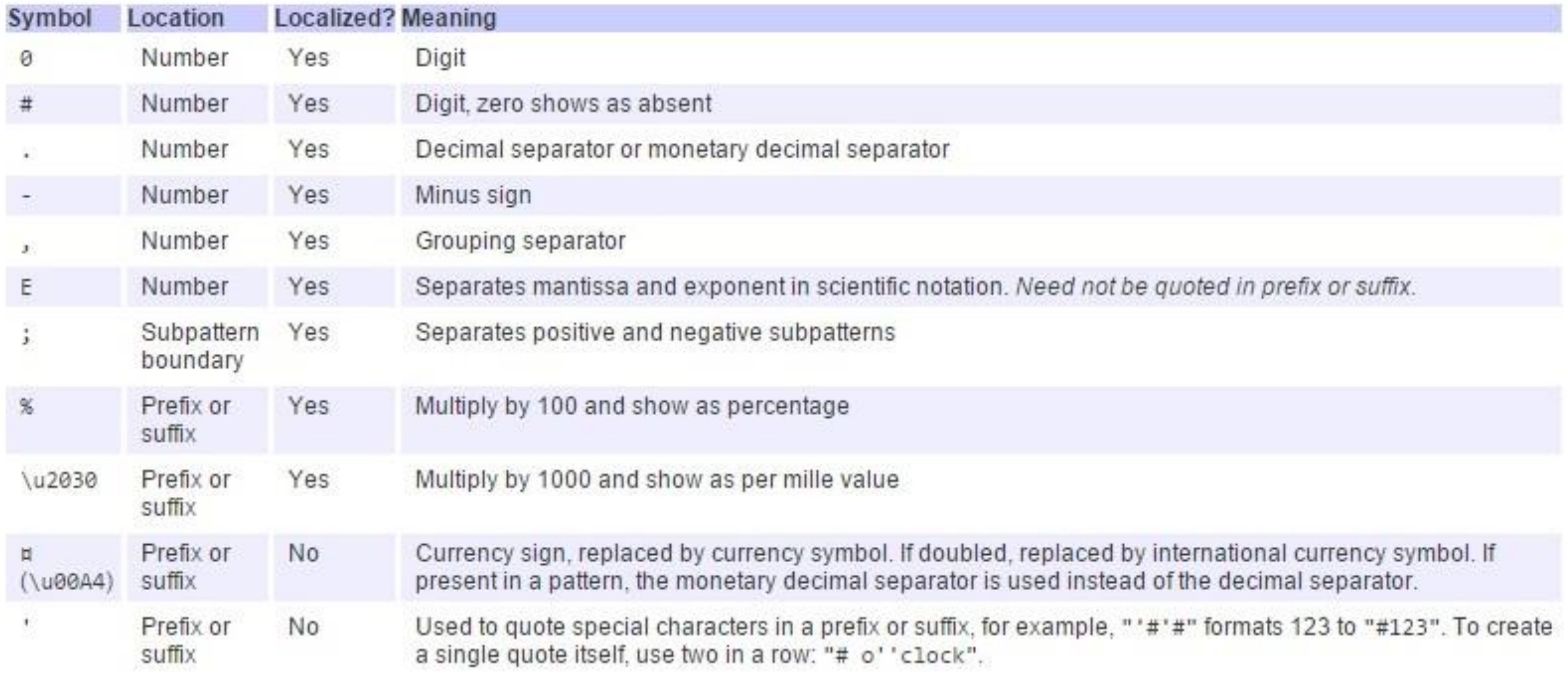
// Equality check.

if (df.format(this.r).equals(df.format(cc.r)) && df.format(this.i).equals(df.format(cc.i))) result=true;

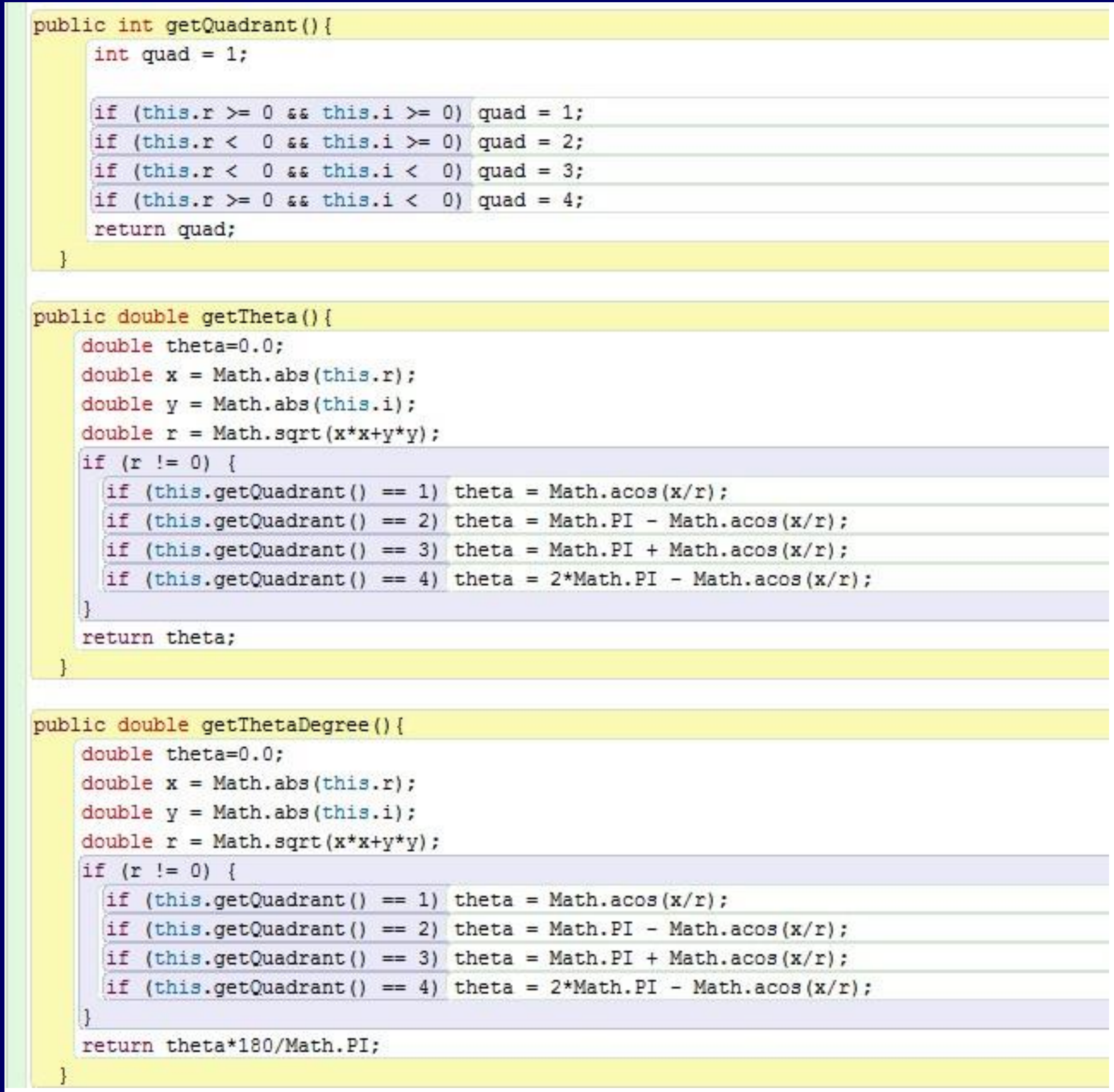
// format: conversion to string. Very powerful when combined with Integer.parseInt(), Double.parstDouble() **This is more systematic than using ((int) (f*10000)) / 10000.0. for double data type precision control.**

Meta-Symbols for DecimalFormat

import java.text.DecimalFormat;



Symbol	Location	Localized?	Meaning
0	Number	Yes	Digit
#	Number	Yes	Digit, zero shows as absent
.	Number	Yes	Decimal separator or monetary decimal separator
-	Number	Yes	Minus sign
,	Number	Yes	Grouping separator
E	Number	Yes	Separates mantissa and exponent in scientific notation. *Need not be quoted in prefix or suffix.*
;	Subpattern boundary	Yes	Separates positive and negative subpatterns
%	Prefix or suffix	Yes	Multiply by 100 and show as percentage
\u2030	Prefix or suffix	Yes	Multiply by 1000 and show as per mille value
¤ (\u00A4)	Prefix or suffix	No	Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator.
'	Prefix or suffix	No	Used to quote special characters in a prefix or suffix, for example, `"'#'#"` formats 123 to `"#123"`. To create a single quote itself, use two in a row: `"# o''clock"`.



```
public int getQuadrant(){
    int quad = 1;

    if (this.r >= 0 && this.i >= 0) quad = 1;
    if (this.r <  0 && this.i >= 0) quad = 2;
    if (this.r <  0 && this.i <  0) quad = 3;
    if (this.r >= 0 && this.i <  0) quad = 4;
    return quad;
  }

public double getTheta(){
    double theta=0.0;
    double x = Math.abs(this.r);
    double y = Math.abs(this.i);
    double r = Math.sqrt(x*x+y*y);
    if (r != 0) {
      if (this.getQuadrant() == 1) theta = Math.acos(x/r);
      if (this.getQuadrant() == 2) theta = Math.PI - Math.acos(x/r);
      if (this.getQuadrant() == 3) theta = Math.PI + Math.acos(x/r);
      if (this.getQuadrant() == 4) theta = 2*Math.PI - Math.acos(x/r);
    }
    return theta;
  }

public double getThetaDegree(){
    double theta=0.0;
    double x = Math.abs(this.r);
    double y = Math.abs(this.i);
    double r = Math.sqrt(x*x+y*y);
    if (r != 0) {
      if (this.getQuadrant() == 1) theta = Math.acos(x/r);
      if (this.getQuadrant() == 2) theta = Math.PI - Math.acos(x/r);
      if (this.getQuadrant() == 3) theta = Math.PI + Math.acos(x/r);
      if (this.getQuadrant() == 4) theta = 2*Math.PI - Math.acos(x/r);
    }
    return theta*180/Math.PI;
  }
```

Why use package?

(1) Organizing code. Put classes of similar function together (same mission). For example, for Washington High school, a package for school administration, a package for student records, a package for course works, a package for school financial management and etc.

(2) Version control. Classes of different implementation (version) can all be tested. **(We use this idea in this chapter 9.)**

(3) Imported module from other sources.

(4) Put code from different developer into a package (for project safety. Quarantine some bad code sometimes). Change package name after integration.

Demo Program:

Complex Package/Complex Class

a1.Complex.java

a1.ComplexTest.java

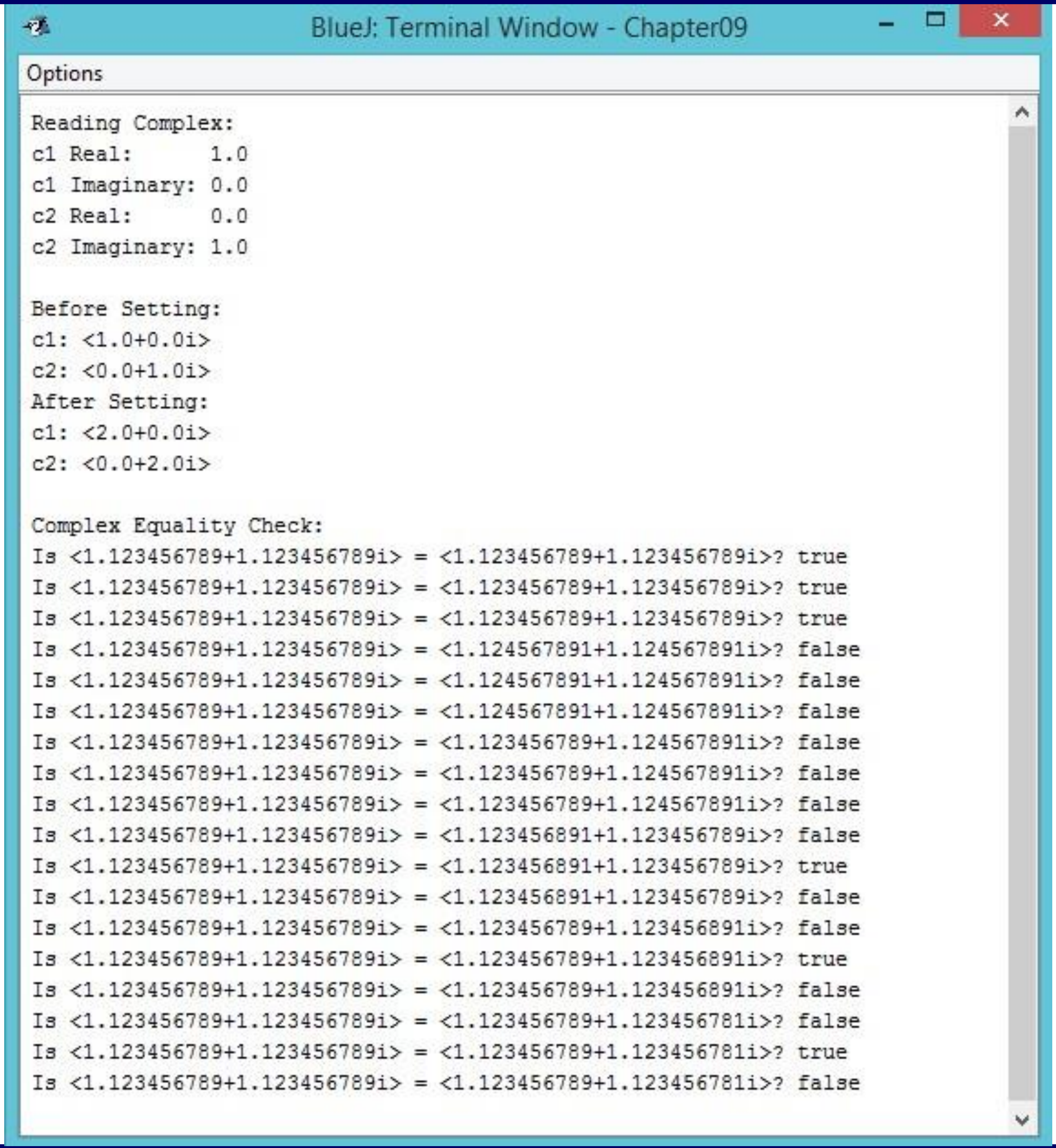
Go BlueJ !!!

Demonstration Program

COMPLEX PACKAGE/COMPLEX CLASS

A1.COMPLEX.JAVA

A1.COMPLEXTEST.JAVA



```
Reading Complex:
c1 Real:      1.0
c1 Imaginary: 0.0
c2 Real:      0.0
c2 Imaginary: 1.0

Before Setting:
c1: <1.0+0.0i>
c2: <0.0+1.0i>
After Setting:
c1: <2.0+0.0i>
c2: <0.0+2.0i>

Complex Equality Check:
Is <1.123456789+1.123456789i> = <1.123456789+1.123456789i>? true
Is <1.123456789+1.123456789i> = <1.123456789+1.123456789i>? true
Is <1.123456789+1.123456789i> = <1.123456789+1.123456789i>? true
Is <1.123456789+1.123456789i> = <1.124567891+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.124567891+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.124567891+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.123456891+1.123456789i>? false
Is <1.123456789+1.123456789i> = <1.123456891+1.123456789i>? true
Is <1.123456789+1.123456789i> = <1.123456891+1.123456789i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.123456891i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.123456891i>? true
Is <1.123456789+1.123456789i> = <1.123456789+1.123456891i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.123456781i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.123456781i>? true
Is <1.123456789+1.123456789i> = <1.123456789+1.123456781i>? false
```

Overview of Class and Objects from Program Structure Point of View

LECTURE 4

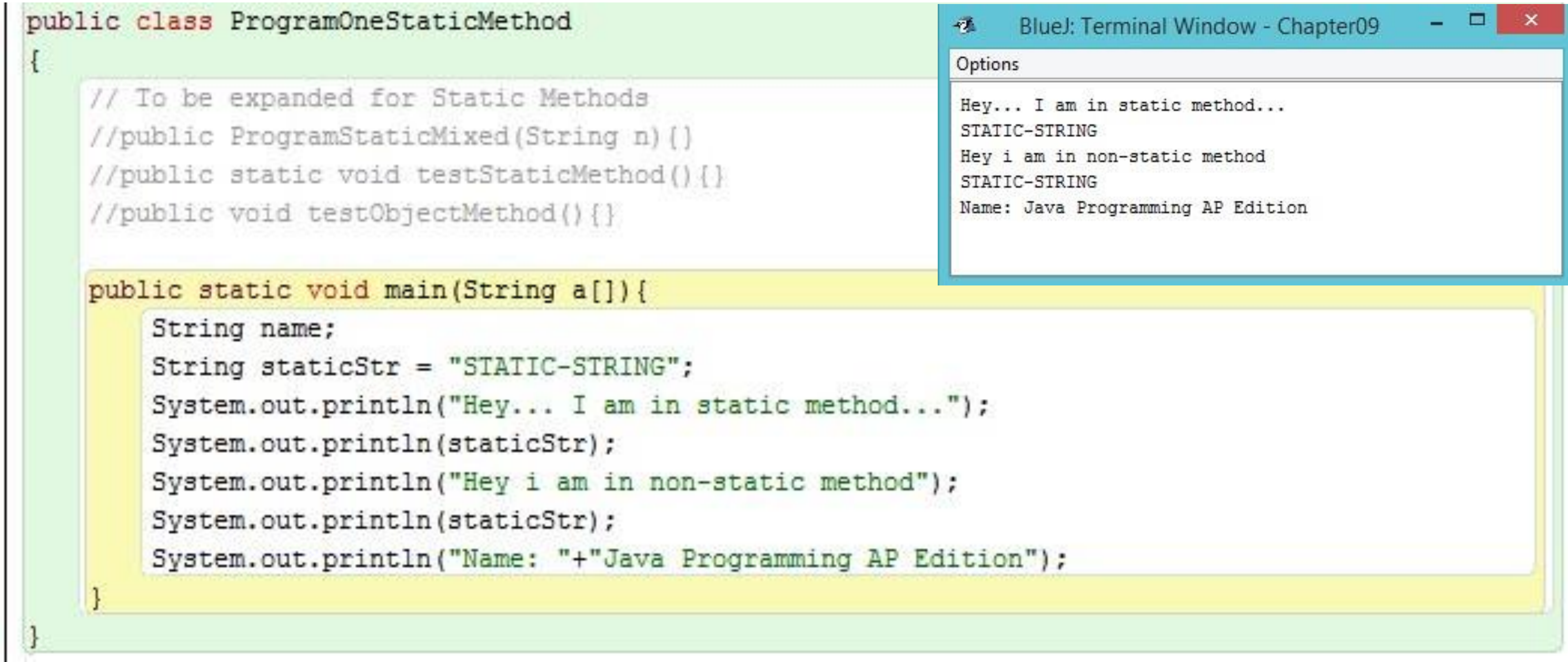
One Static Method Program

ProgramOneStaticMethod.java

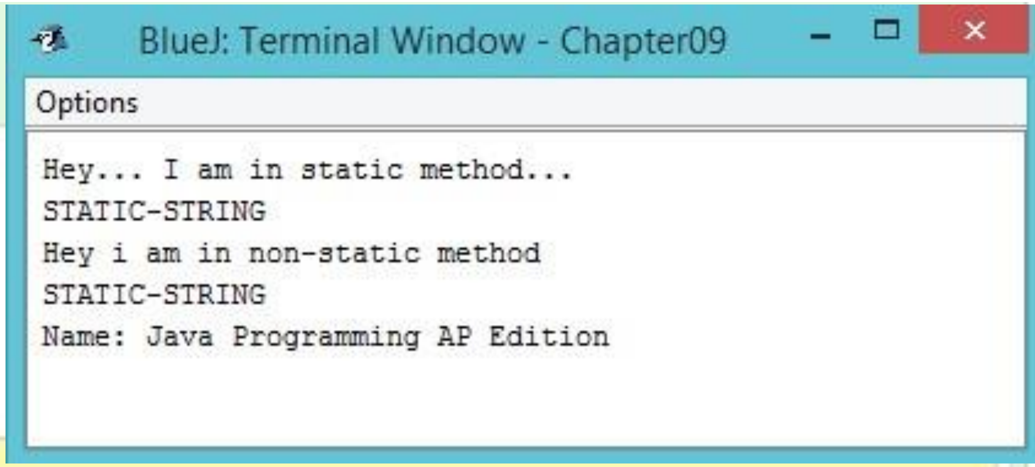
- One Method: public static void main(String[] args) {…}
- All variable defined at the beginning of the main method.
- Simple program. No calling other methods. (Elementary Programming: Sequential Programming)

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
```

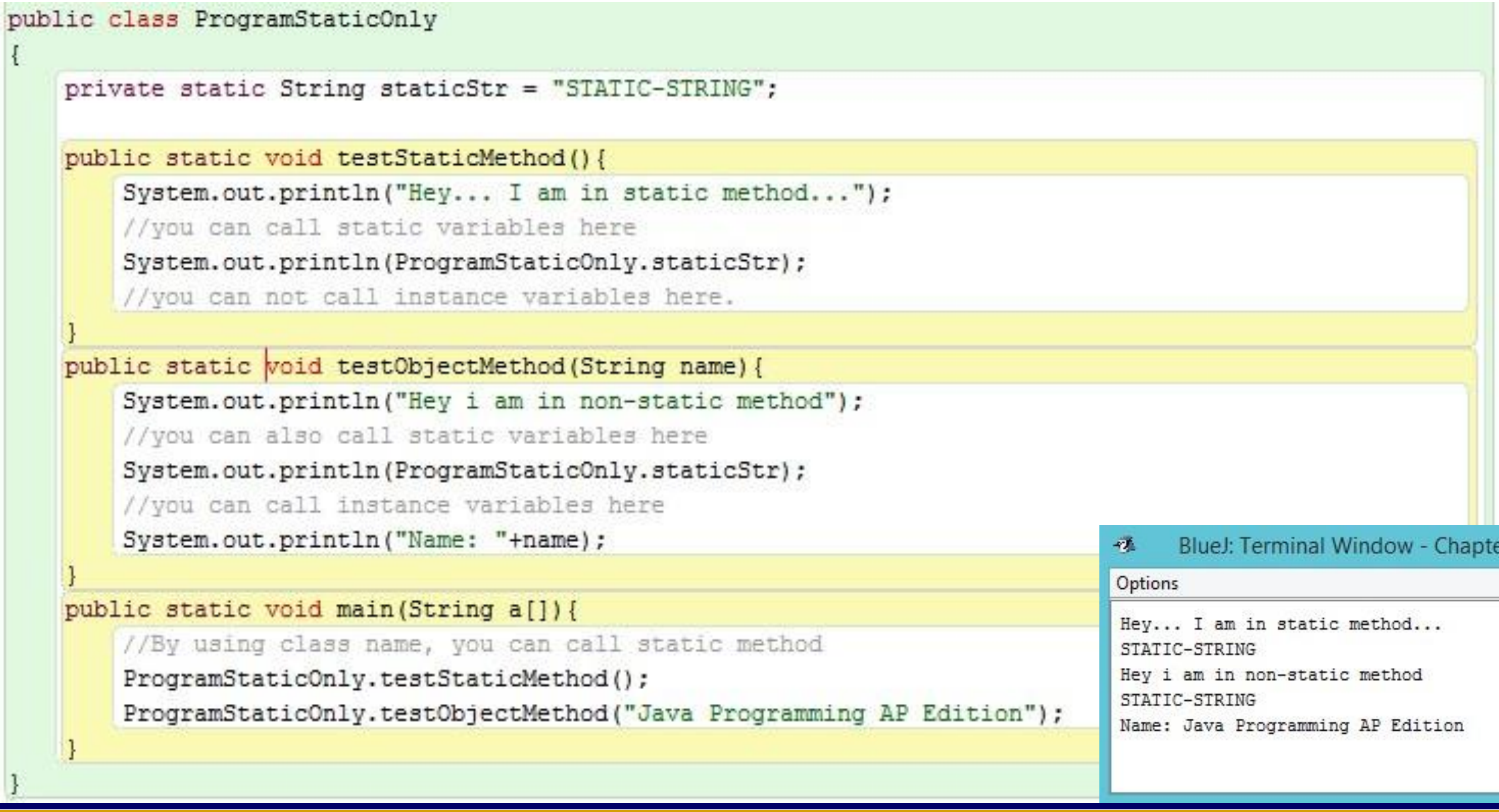
Static Method Only Program Structures

(ProgramStaticOnly.java Equivalent to Structural Programming)

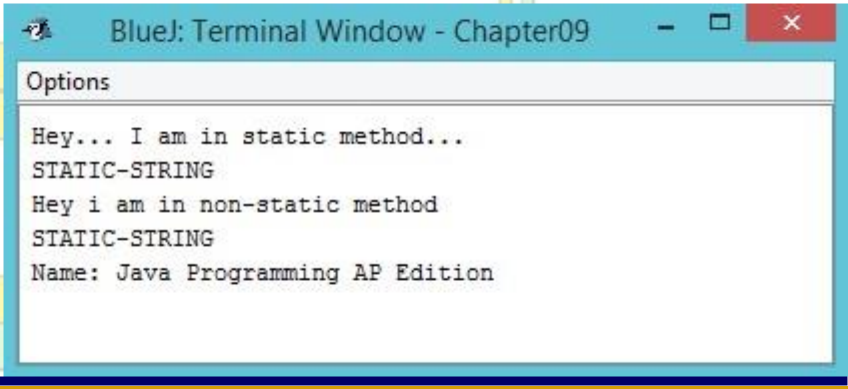
- Global variables defined at the class properties declaration region as static variables.
- Local variables defined at the main method or other methods (or code blocks)
- Use ClassName.method() to call static methods or Use method() to call the methods (if only one class).
- Multiple class, you can 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)



```java
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
```

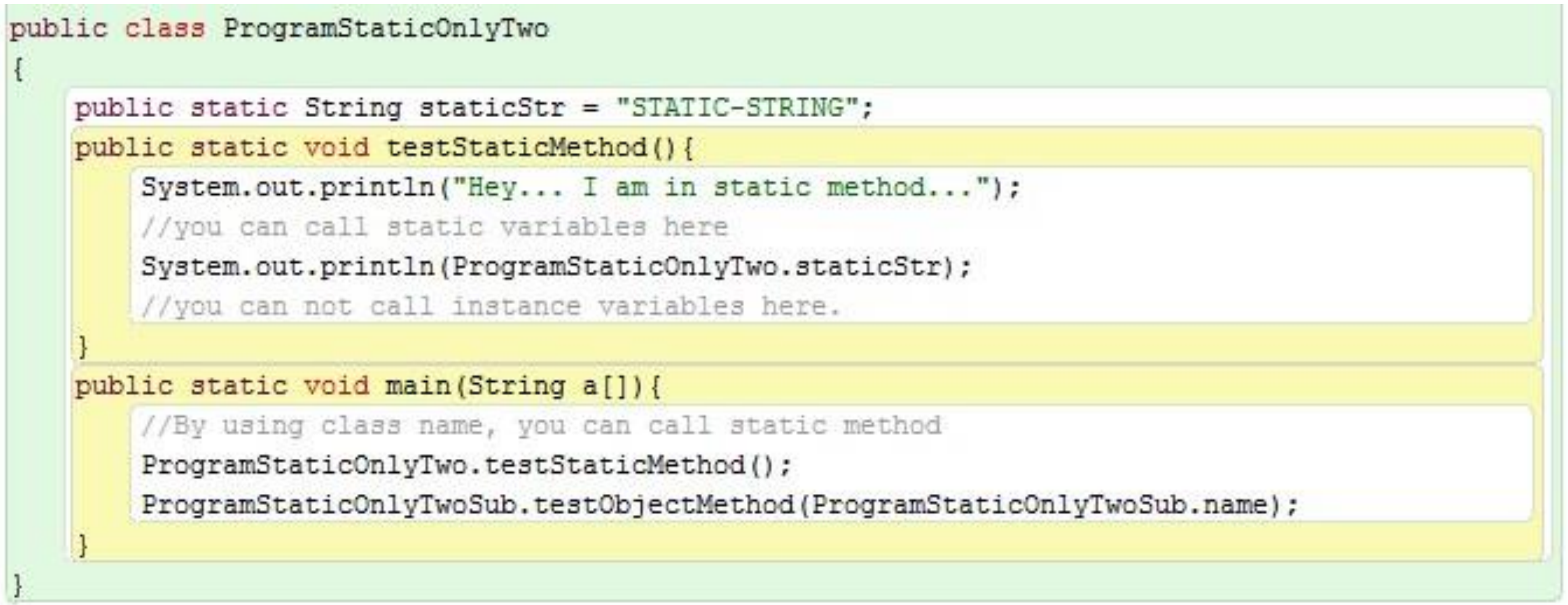
Static Method Only with Two Classes

(ProgramStaticOnlyTwo.java Equivalent to Structural Programming)

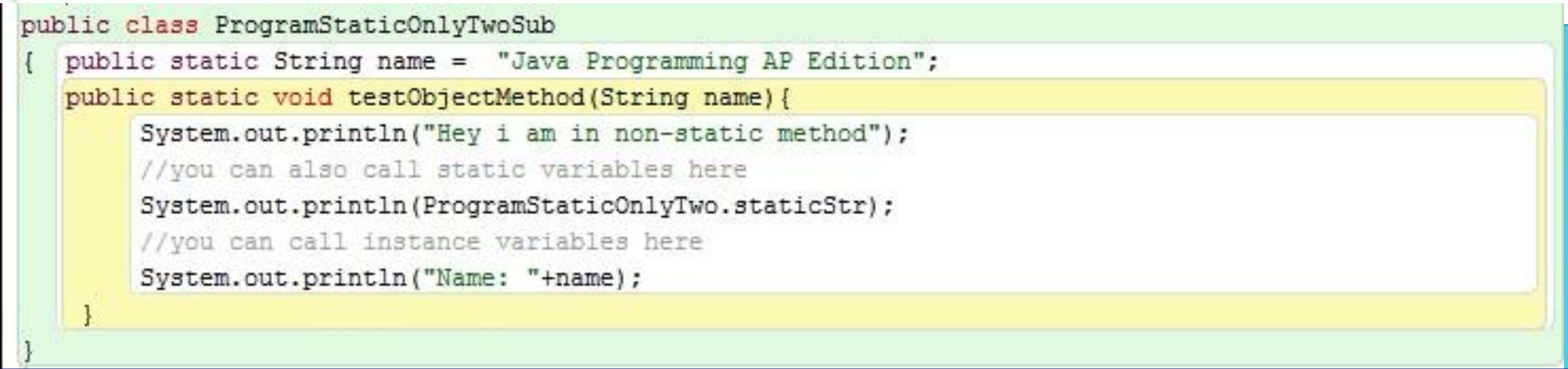
- Similar to Static Method Only Program except that two classes are used. (One is considered as an external program call.
- Because the two classes are in the same package, there is no need to import. If they are in different package, then the visibility modifiers need to be specified as public.
- Similar 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

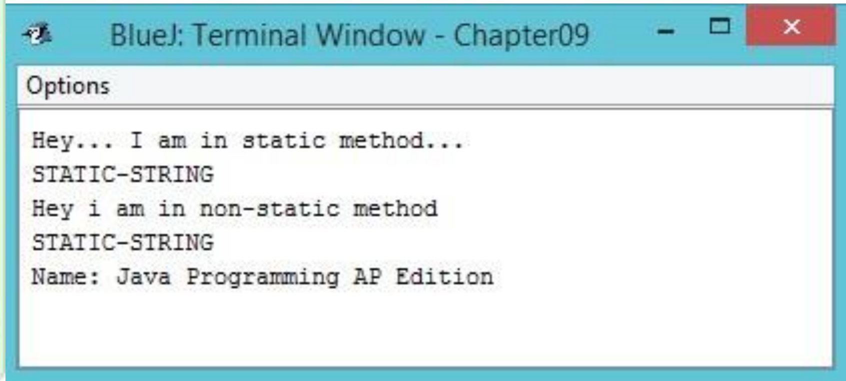
(ProgramStaticOnlyTwo.java Equivalent to Structural Programming)



```
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);
    }
}
```



```
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);
    }
}
```



BlueJ: Terminal Window - Chapter09

Options

```
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:

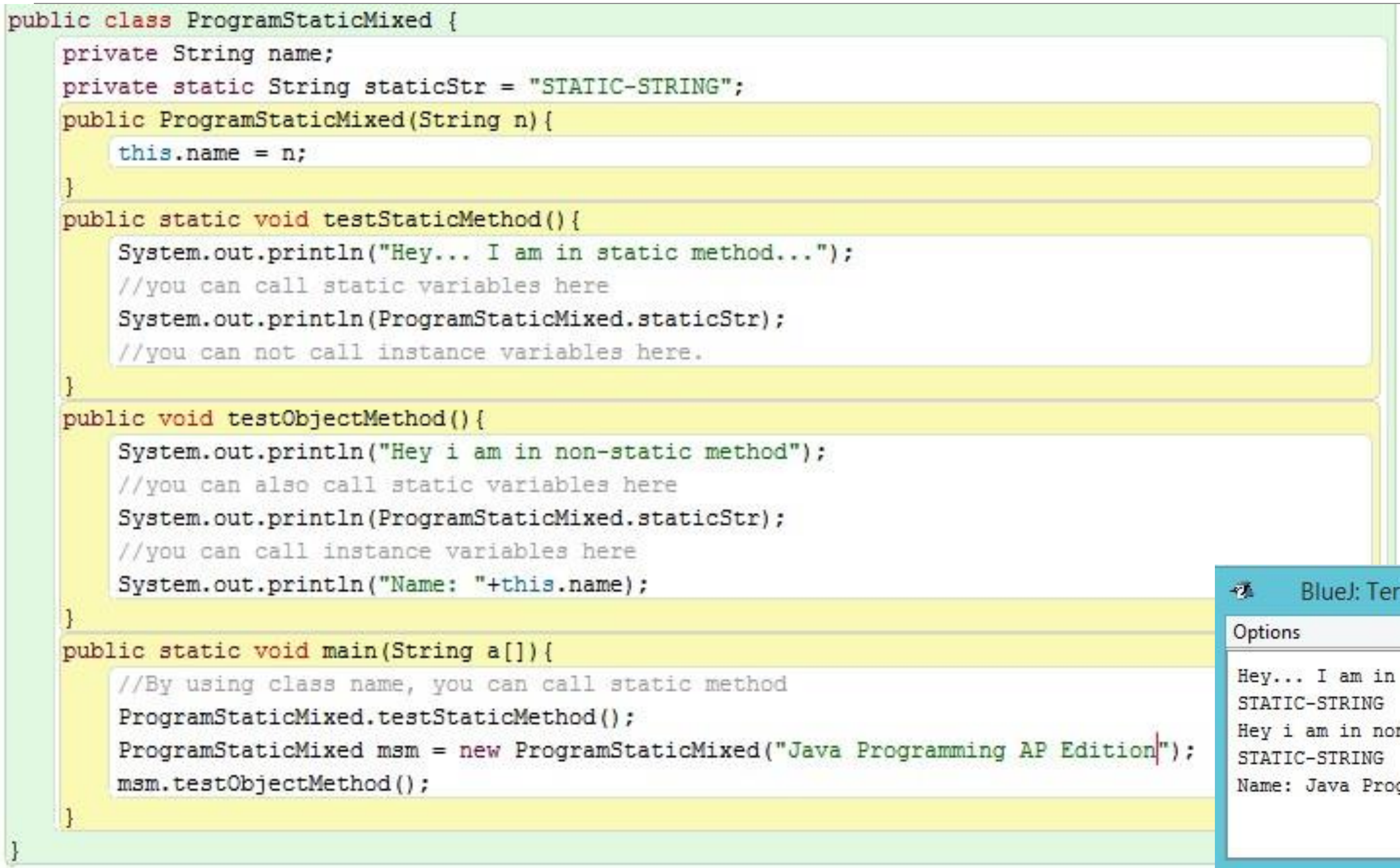
Static variables are belongs to the class and not to the object. These are only once, at the starting of the execution. Static variables are not part of object state, means there is only one copy of the values will be served to all instances. You can call static variable with reference to class name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

Java static methods:

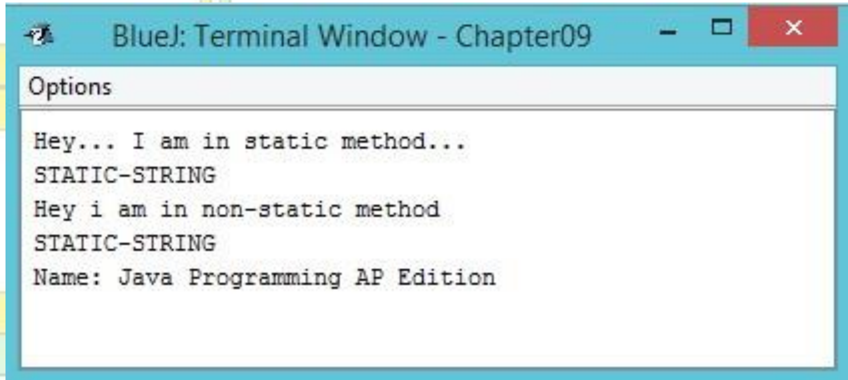
Static methods are also similar to static variables, you can access them with reference to class name, without creating object. Inside static methods, you cannot access instance variables or instance methods. You can only access static variables or static methods.

Program Structure Static Mixed Style

ProgramStaticMixed.java



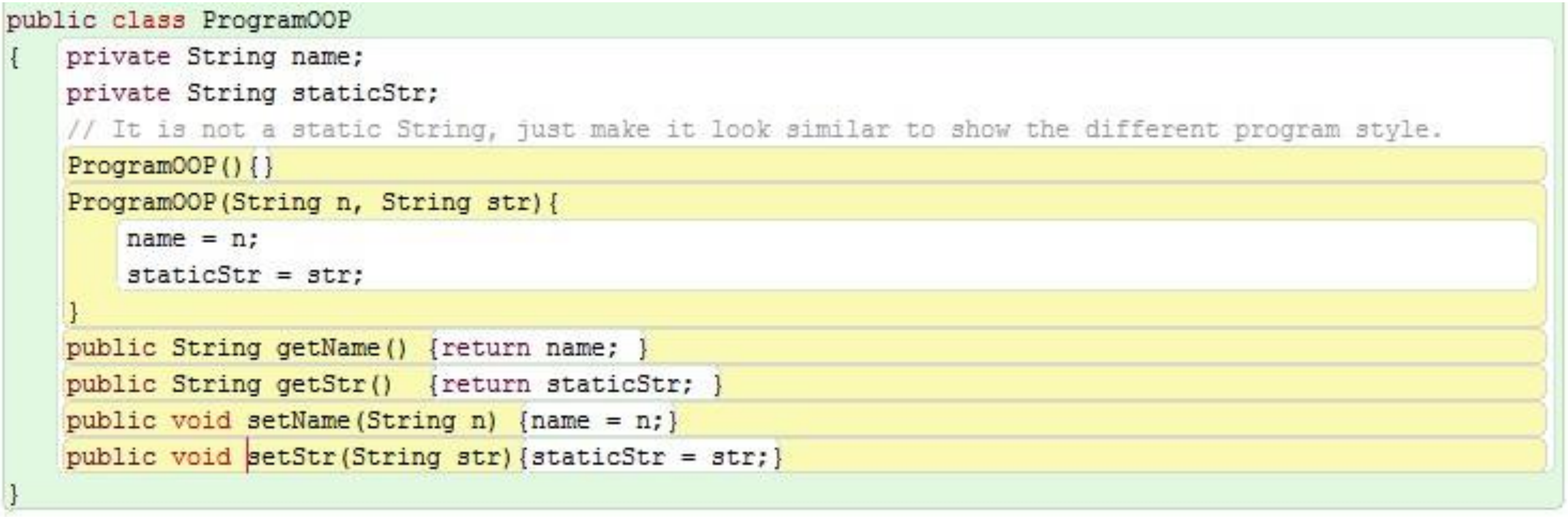
```java
public class ProgramStaticMixed {
    private String name;
    private static String staticStr = "STATIC-STRING";
    public ProgramStaticMixed(String n){
        this.name = n;
    }
    public static void testStaticMethod(){
        System.out.println("Hey... I am in static method...");
        //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)

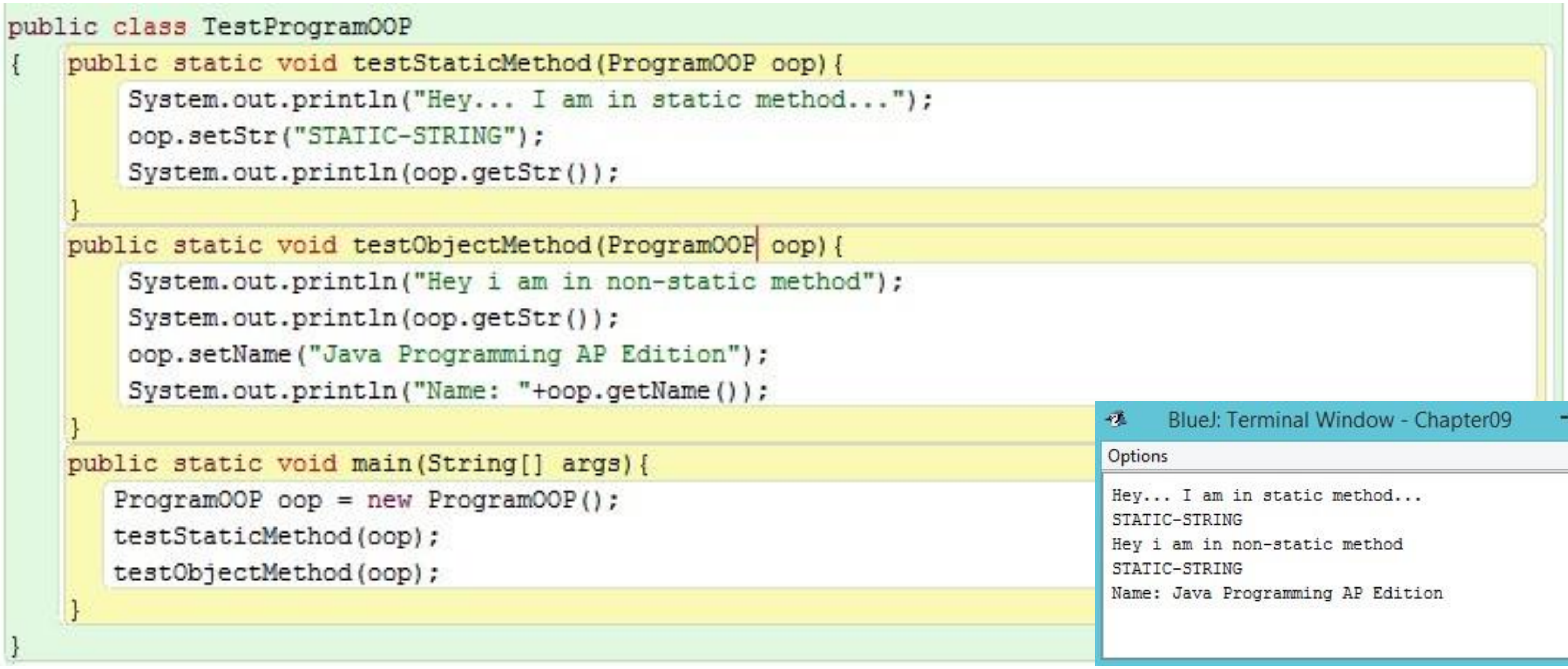
ProgramObjectOriented.java



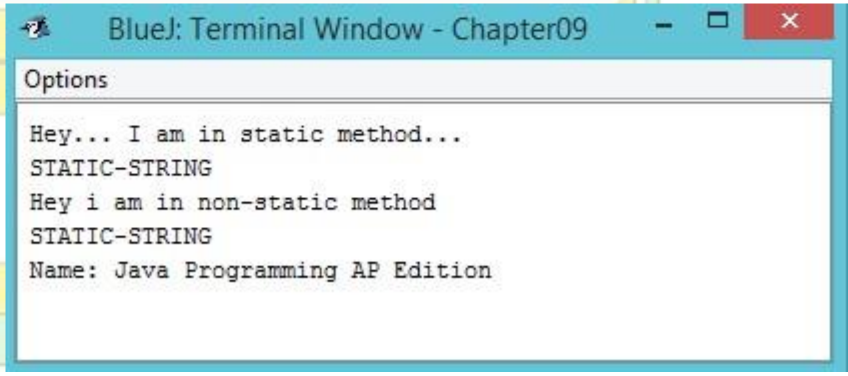
```
public class ProgramOOP
{   private String name;
    private String staticStr;
    // It is not a static String, just make it look similar to show the different program style.
    ProgramOOP(){}
    ProgramOOP(String n, String str){
        name = n;
        staticStr = str;
    }
    public String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

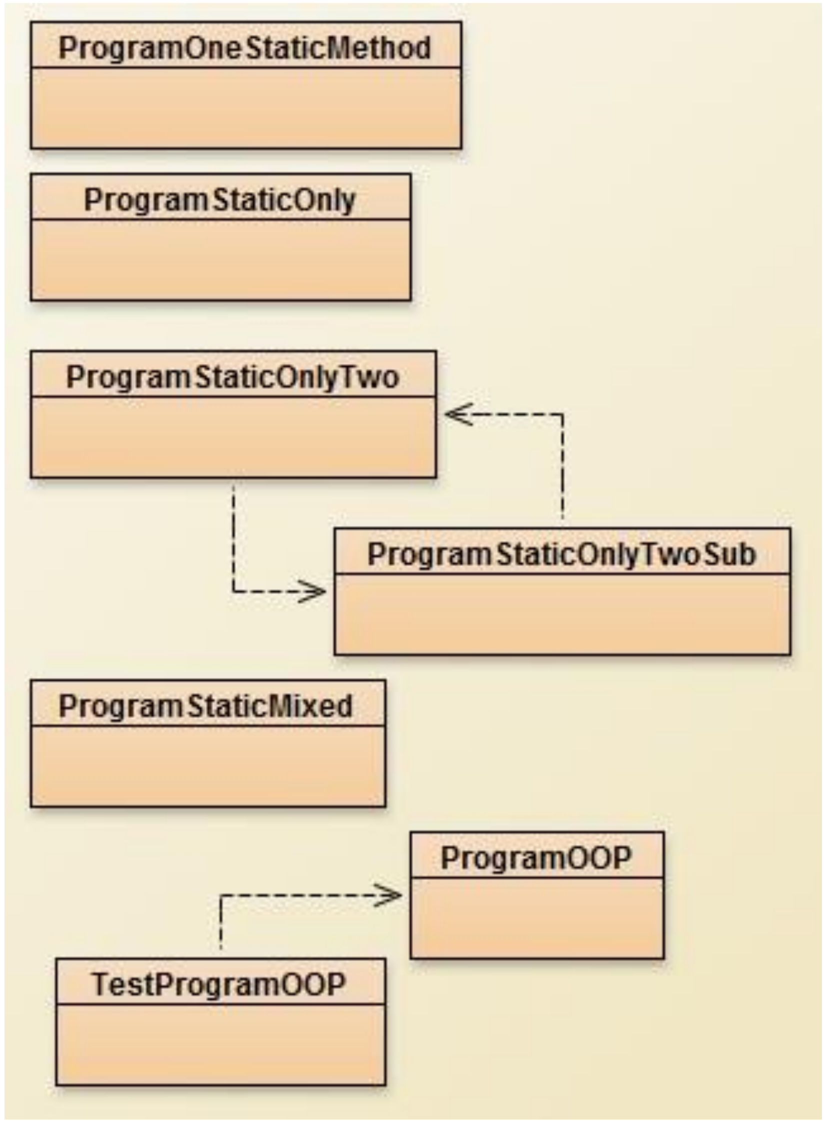
TestObject-Oriented.java



```
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

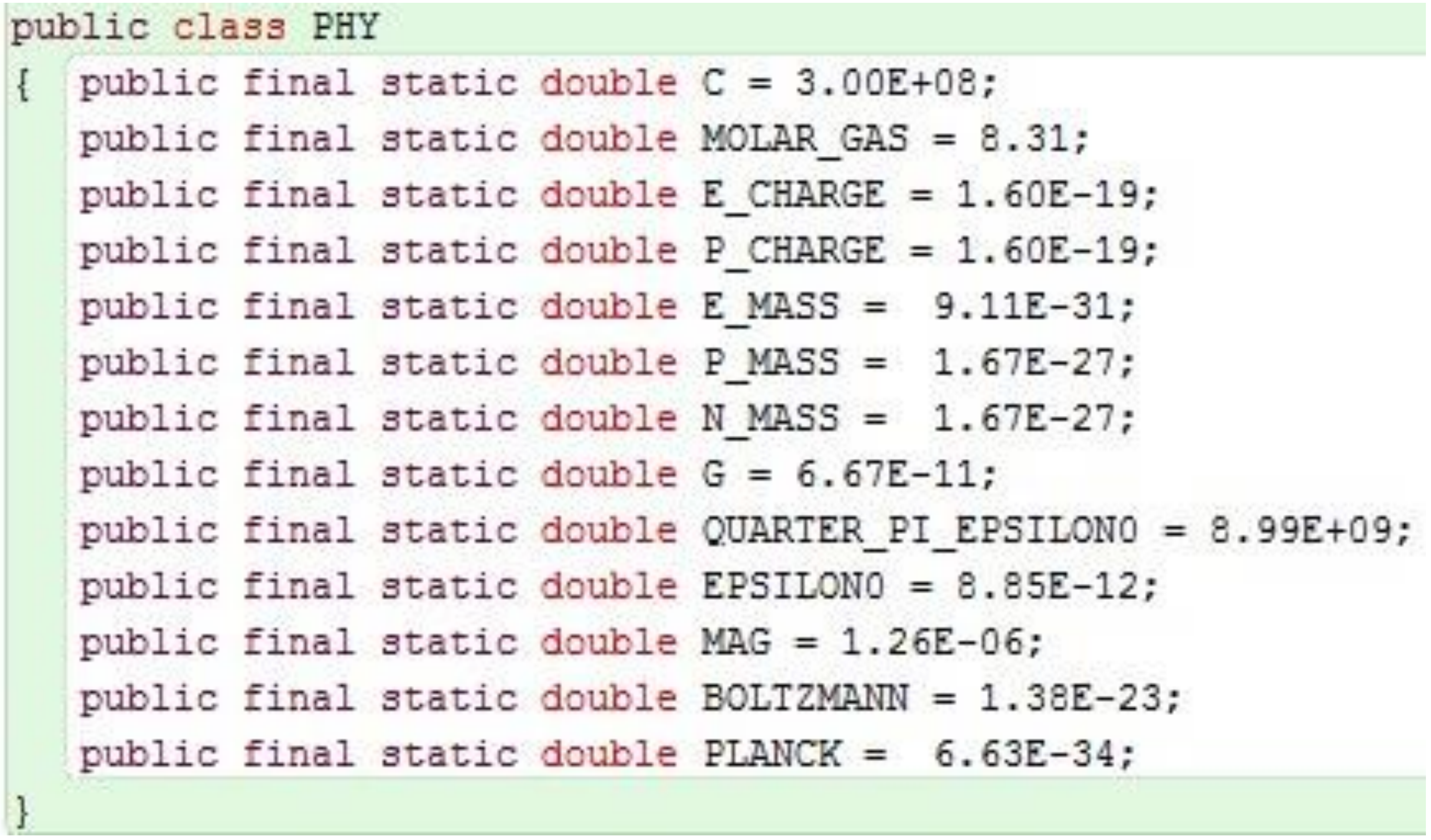
STATIC PROGRAM SERIES

Overview of Class and Objects from Data Structure Point of Views

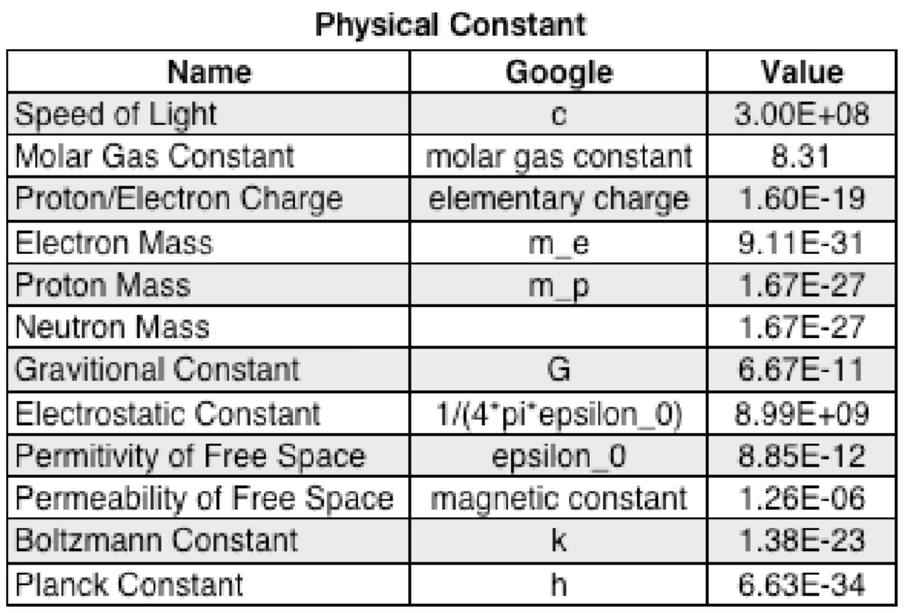
LECTURE 5

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;
}
```



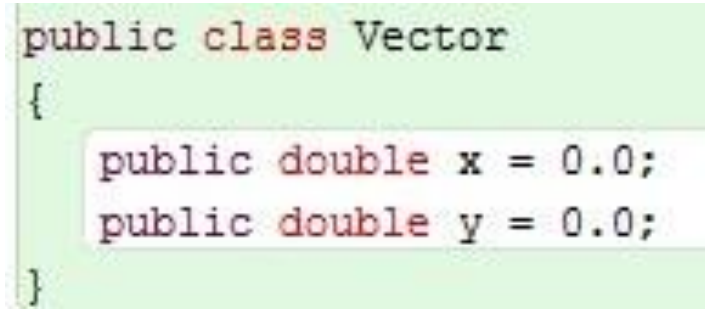
Physical Constant

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

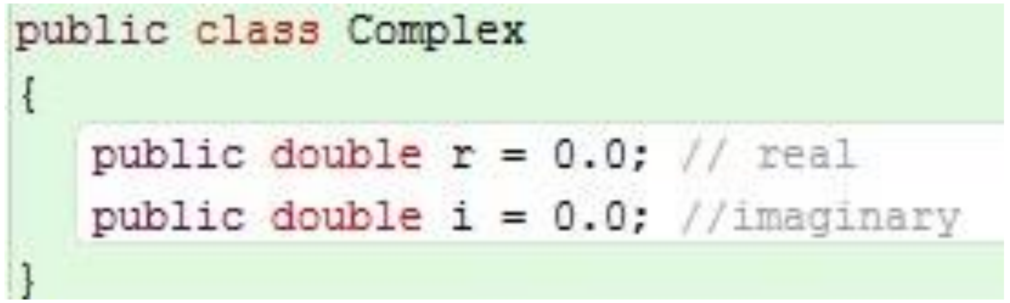
Use **PHY.C** for Speed of Light

Data Vector of Multiple-Tuple Class

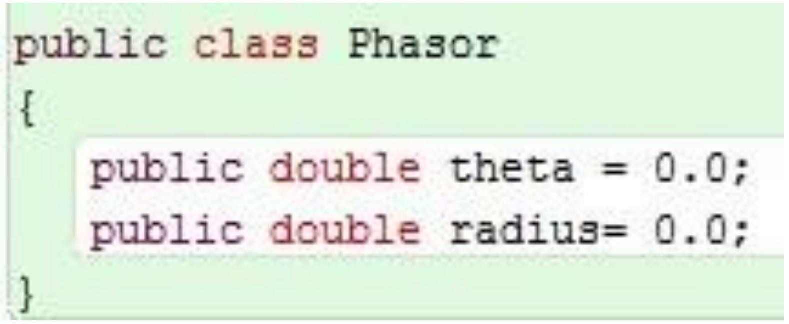
2D-Vector, Complex Number and Phasor



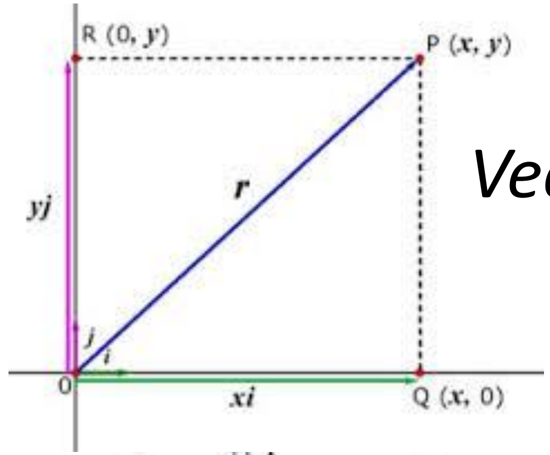
```java
public class Vector
{
    public double x = 0.0;
    public double y = 0.0;
}
```



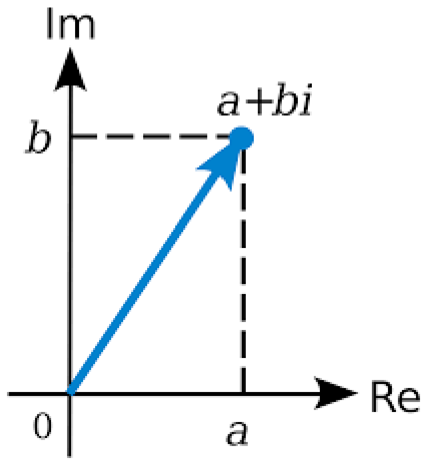
```java
public class Complex
{
    public double r = 0.0; // real
    public double i = 0.0; //imaginary
}
```



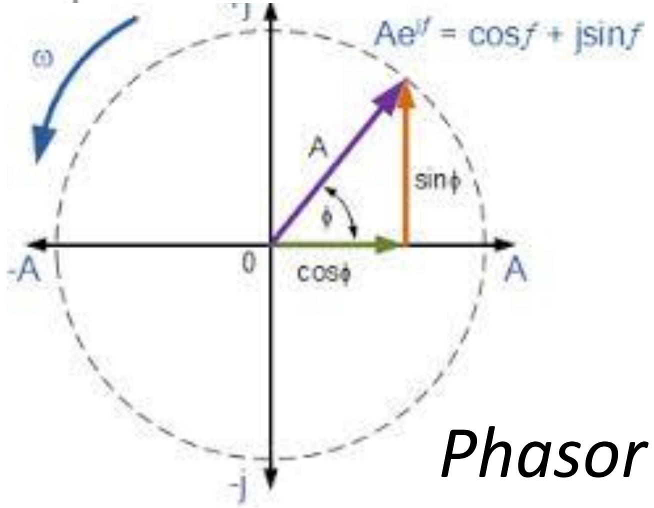
```java
public class Phasor
{
    public double theta = 0.0;
    public double radius= 0.0;
}
```



Vector



Complex



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;
}
```

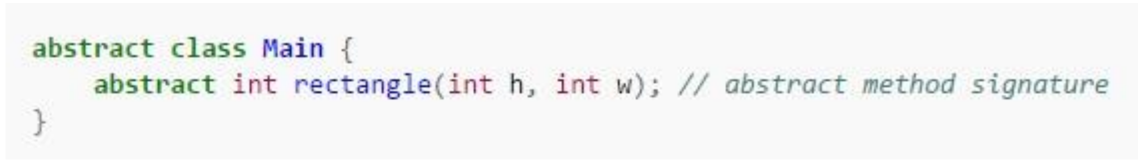
Methods

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:



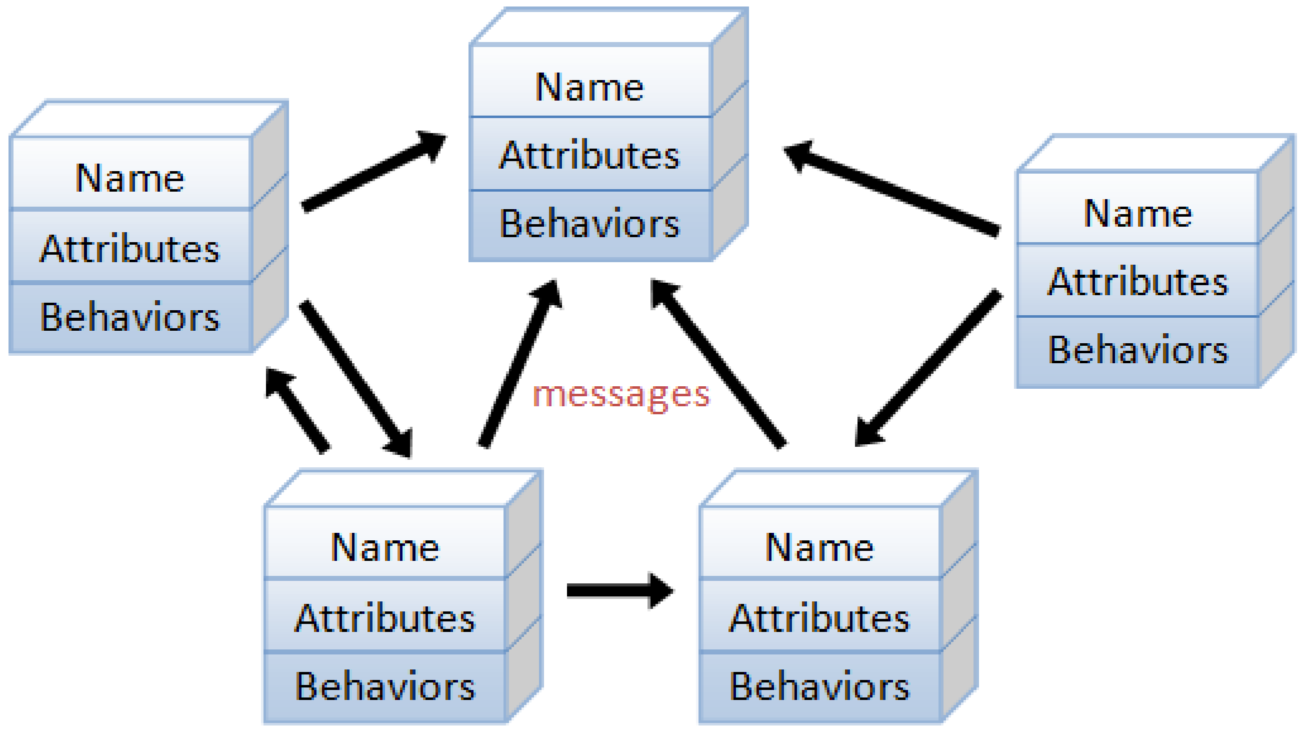
```
abstract class Main {
    abstract int rectangle(int h, int w); // abstract method signature
}
```

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

Lab:

add abs(), add(), minus(), multiply(), divide() instance methods to Complex Class

LECTURE 6

Instance Methods

Use of Instance Methods:

Instance Method is a method belonging to an object not the class. It is available after the object is created.

object.instanceMethod(parameter);

Example:

Complex c1 = new Complex(1, 2);

Complex c2 = new Complex(2, 3);

c1.add(c2);

Complex Addition and Subtraction

COMPLEX NUMBERS

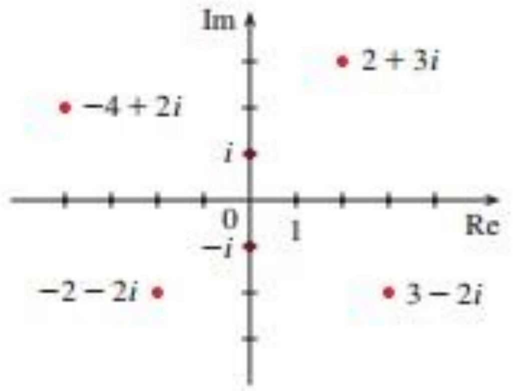


FIGURE 1
Complex numbers as points in the Argand plane

A **complex number** can be represented by an expression of the form $a + bi$, where a and b are real numbers and i is a symbol with the property that $i^2 = -1$. The complex number $a + bi$ can also be represented by the ordered pair (a, b) and plotted as a point in a plane (called the Argand plane) as in Figure 1. Thus, the complex number $i = 0 + 1 \cdot i$ is identified with the point $(0, 1)$.

The **real part** of the complex number $a + bi$ is the real number a and the **imaginary part** is the real number b. Thus, the real part of $4 - 3i$ is 4 and the imaginary part is -3. Two complex numbers $a + bi$ and $c + di$ are **equal** if $a = c$ and $b = d$, that is, their real parts are equal and their imaginary parts are equal. In the Argand plane the horizontal axis is called the real axis and the vertical axis is called the imaginary axis.

The sum and difference of two complex numbers are defined by adding or subtracting their real parts and their imaginary parts:

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

$$(a + bi) - (c + di) = (a - c) + (b - d)i$$

Conjugate

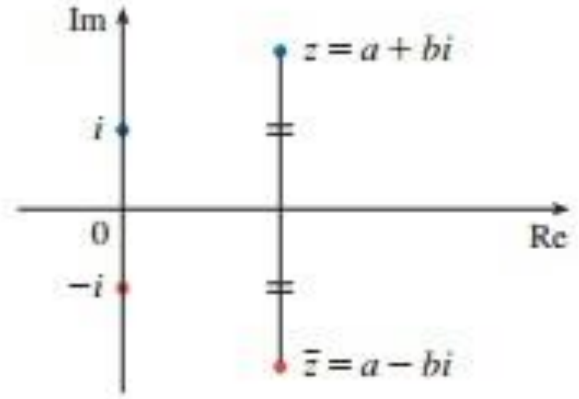


FIGURE 2

The geometric interpretation of the complex conjugate is shown in Figure 2: $\bar{z}$ is the reflection of z in the real axis. We list some of the properties of the complex conjugate in the following box. The proofs follow from the definition and are requested in Exercise 18.

Properties of Conjugates

$$\overline{z + w} = \bar{z} + \bar{w} \qquad \overline{zw} = \bar{z}\,\bar{w} \qquad \overline{z^n} = \bar{z}^n$$

Absolute Value of Complex Numbers

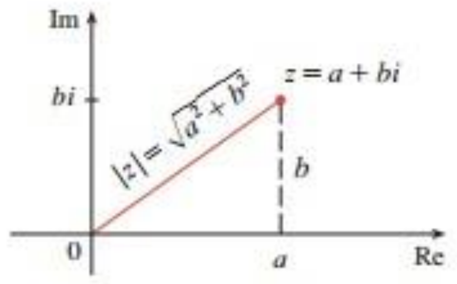


FIGURE 3

The **modulus**, or **absolute value**, $|z|$ of a complex number $z = a + bi$ is its distance from the origin. From Figure 3 we see that if $z = a + bi$, then

$$\boxed{|z| = \sqrt{a^2 + b^2}}$$

Notice that

$$z\bar{z} = (a + bi)(a - bi) = a^2 + abi - abi - b^2 i^2 = a^2 + b^2$$

and so

$$\boxed{z\bar{z} = |z|^2}$$

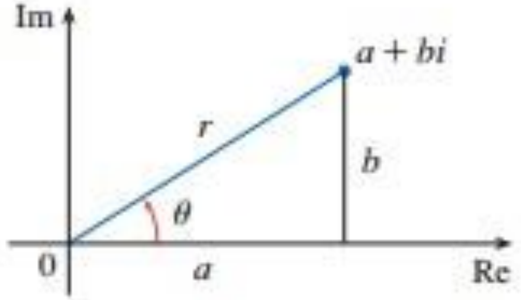
This explains why the division procedure in Example 2 works in general:

$$\frac{z}{w} = \frac{z\bar{w}}{w\bar{w}} = \frac{z\bar{w}}{|w|^2}$$

Since $i^2 = -1$, we can think of i as a square root of -1. But notice that we also have $(-i)^2 = i^2 = -1$ and so $-i$ is also a square root of -1. We say that i is the **principal square root** of -1 and write $\sqrt{-1} = i$. In general, if c is any positive number, we write

$$\sqrt{-c} = \sqrt{c}\, i$$

POLAR FORM



We know that any complex number $z = a + bi$ can be considered as a point (a, b) and that any such point can be represented by polar coordinates (r, θ) with $r \geqslant 0$. In fact,

$$a = r\cos\theta \qquad b = r\sin\theta$$

as in Figure 4. Therefore, we have

Thus, we can write any complex number z in the form

$$\boxed{z = r(\cos\theta + i\sin\theta)}$$

where $$r = |z| = \sqrt{a^2 + b^2} \qquad \text{and} \qquad \tan\theta = \frac{b}{a}$$

The angle θ is called the **argument** of z and we write $\theta = \arg(z)$. Note that $\arg(z)$ is not unique; any two arguments of z differ by an integer multiple of 2π.

eC Learning Channel

Complex Multiplication

The polar form of complex numbers gives insight into multiplication and division. Let

$$z_1 = r_1(\cos\theta_1 + i\sin\theta_1) \qquad z_2 = r_2(\cos\theta_2 + i\sin\theta_2)$$

be two complex numbers written in polar form. Then

$$\begin{aligned} z_1 z_2 &= r_1 r_2(\cos\theta_1 + i\sin\theta_1)(\cos\theta_2 + i\sin\theta_2) \\ &= r_1 r_2[(\cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2) + i(\sin\theta_1\cos\theta_2 + \cos\theta_1\sin\theta_2)] \end{aligned}$$

Therefore, using the addition formulas for cosine and sine, we have

1 $$\boxed{z_1 z_2 = r_1 r_2[\cos(\theta_1 + \theta_2) + i\sin(\theta_1 + \theta_2)]}$$

This formula says that *to multiply two complex numbers we multiply the moduli and add the arguments.* (See Figure 6.)

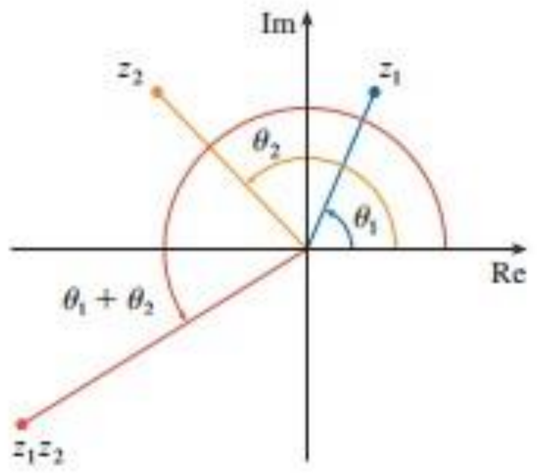


FIGURE 6

Complex Division

A similar argument using the subtraction formulas for sine and cosine shows that *to divide two complex numbers we divide the moduli and subtract the arguments.*

$$\boxed{\frac{z_1}{z_2} = \frac{r_1}{r_2}[\cos(\theta_1 - \theta_2) + i\sin(\theta_1 - \theta_2)] \qquad z_2 \neq 0}$$

In particular, taking $z_1 = 1$ and $z_2 = z$, (and therefore $\theta_1 = 0$ and $\theta_2 = \theta$), we have the following, which is illustrated in Figure 7.

$$\boxed{\text{If} \quad z = r(\cos\theta + i\sin\theta), \quad \text{then} \quad \frac{1}{z} = \frac{1}{r}(\cos\theta - i\sin\theta).}$$

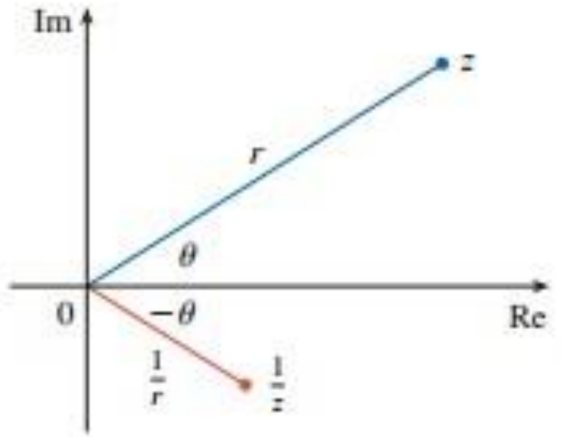


FIGURE 7

COMPLEX EXPONENTIALS

We also need to give a meaning to the expression e^z when $z = x + iy$ is a complex number. The theory of infinite series as developed in Chapter 8 can be extended to the case where the terms are complex numbers. Using the Taylor series for e^x (8.7.12) as our guide, we define

$$\boxed{4} \qquad e^z = \sum_{n=0}^{\infty} \frac{z^n}{n!} = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \cdots$$

and it turns out that this complex exponential function has the same properties as the real exponential function. In particular, it is true that

$$\boxed{5} \qquad e^{z_1+z_2} = e^{z_1}e^{z_2}$$

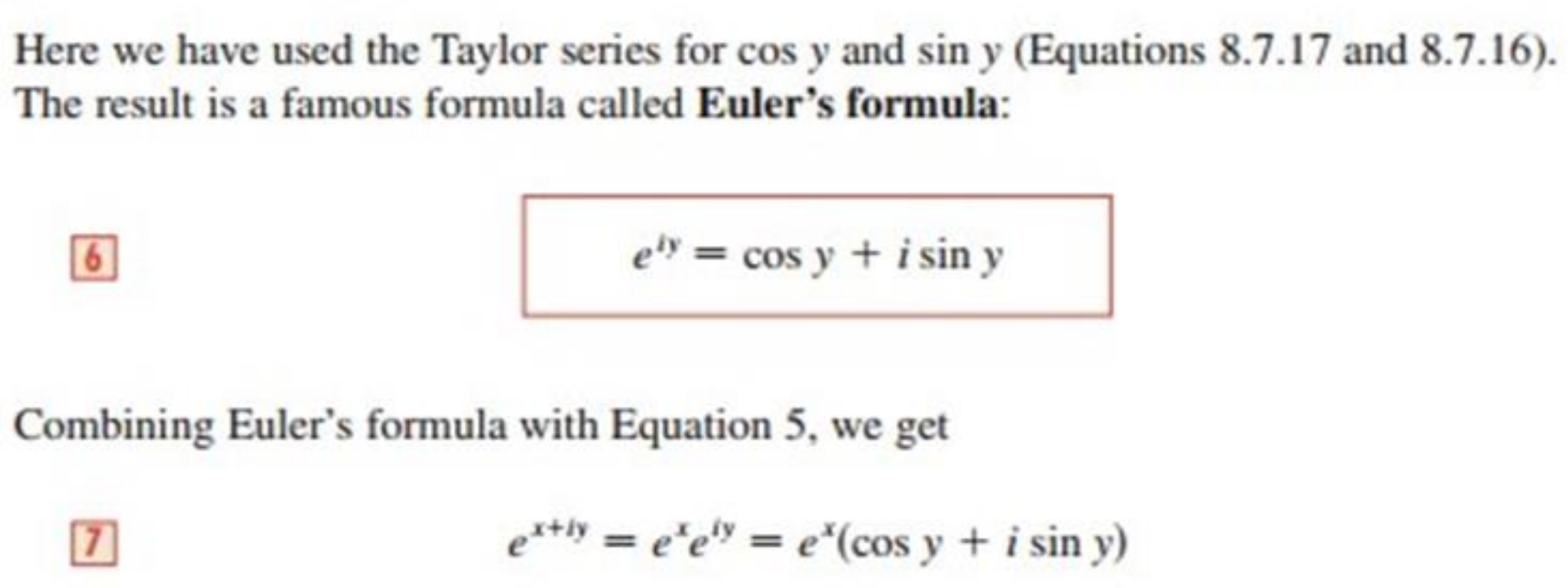
If we put $z = iy$, where y is a real number, in Equation 4, and use the facts that

$$i^2 = -1, \quad i^3 = i^2 i = -i, \quad i^4 = 1, \quad i^5 = i, \quad \ldots$$

we get

$$\begin{aligned} e^{iy} &= 1 + iy + \frac{(iy)^2}{2!} + \frac{(iy)^3}{3!} + \frac{(iy)^4}{4!} + \frac{(iy)^5}{5!} + \cdots \\ &= 1 + iy - \frac{y^2}{2!} - i\frac{y^3}{3!} + \frac{y^4}{4!} + i\frac{y^5}{5!} + \cdots \\ &= \left(1 - \frac{y^2}{2!} + \frac{y^4}{4!} - \frac{y^6}{6!} + \cdots\right) + i\left(y - \frac{y^3}{3!} + \frac{y^5}{5!} - \cdots\right) \\ &= \cos y + i \sin y \end{aligned}$$

Euler’s Formula



Here we have used the Taylor series for cos y and sin y (Equations 8.7.17 and 8.7.16). The result is a famous formula called **Euler’s formula**:

6
$$\boxed{e^{iy} = \cos y + i \sin y}$$

Combining Euler’s formula with Equation 5, we get

7
$$e^{x+iy} = e^x e^{iy} = e^x(\cos y + i \sin y)$$

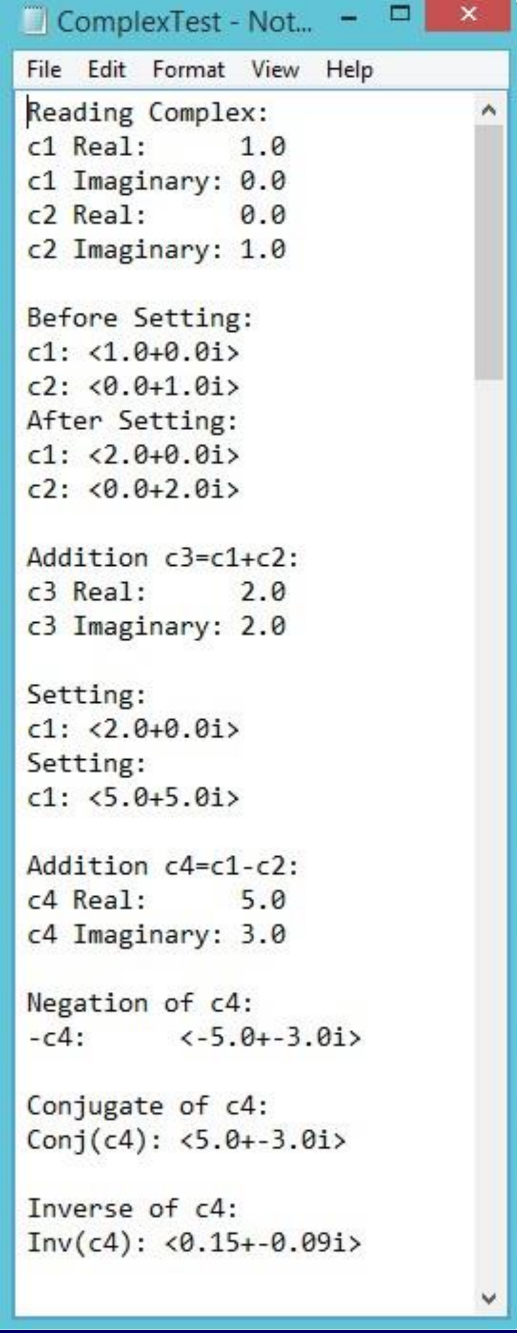
Lab

COMPLEX NUMBERS

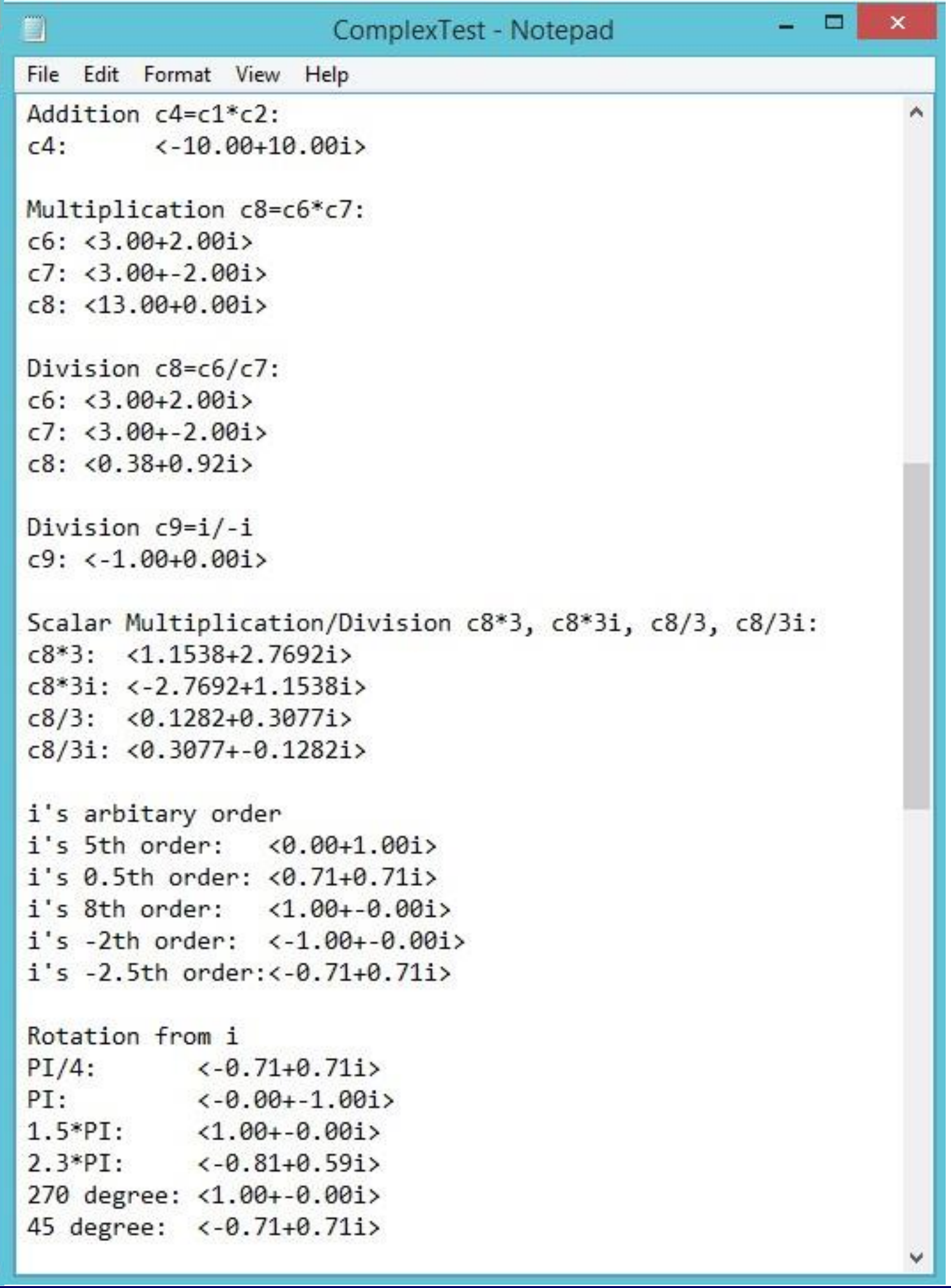
Lab Project:

sample answer a2.zip

- Create an a2 project. Create a Complex class. Copy the content of the Complex Class from a1 package to the Complex.java file in a2 package you just created.
- Write the instance methods (add, minus, multiply, divide, conjugate, abs, pow (if you can)) .
- Also write a **ComplexTest** class to test your own **Complex class**.
- After you finished, download the a2.zip and unzip it to another location to study the sample answer file. Maybe your version is better.



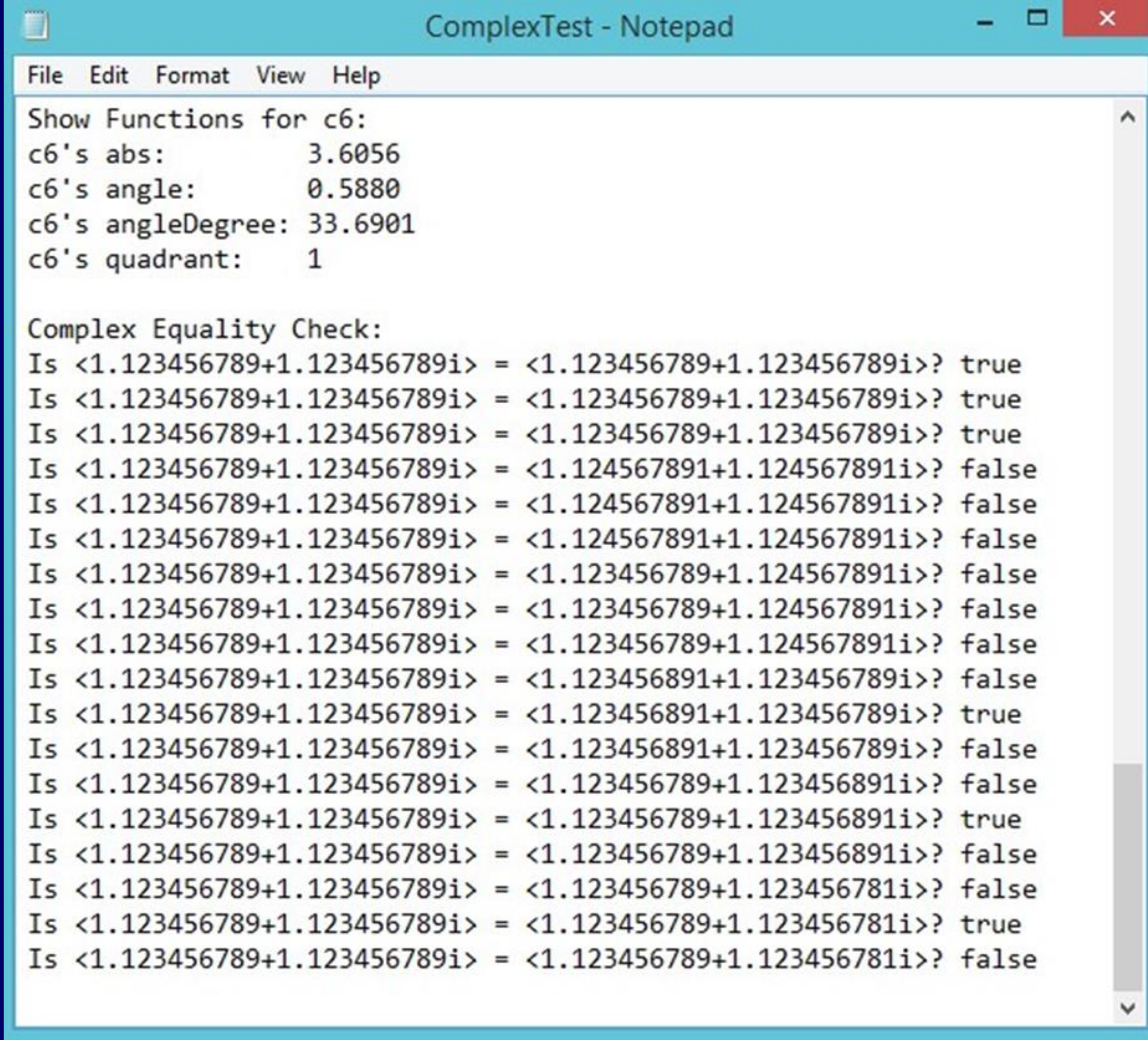
```
Reading Complex:
c1 Real:      1.0
c1 Imaginary: 0.0
c2 Real:      0.0
c2 Imaginary: 1.0

Before Setting:
c1: <1.0+0.0i>
c2: <0.0+1.0i>
After Setting:
c1: <2.0+0.0i>
c2: <0.0+2.0i>

Addition c3=c1+c2:
c3 Real:      2.0
c3 Imaginary: 2.0

Setting:
c1: <2.0+0.0i>
Setting:
c1: <5.0+5.0i>

Addition c4=c1-c2:
c4 Real:      5.0
c4 Imaginary: 3.0

Negation of c4:
-c4:        <-5.0+-3.0i>

Conjugate of c4:
Conj(c4): <5.0+-3.0i>

Inverse of c4:
Inv(c4): <0.15+-0.09i>
```



```
Addition c4=c1*c2:
c4:       <-10.00+10.00i>

Multiplication c8=c6*c7:
c6: <3.00+2.00i>
c7: <3.00+-2.00i>
c8: <13.00+0.00i>

Division c8=c6/c7:
c6: <3.00+2.00i>
c7: <3.00+-2.00i>
c8: <0.38+0.92i>

Division c9=i/-i
c9: <-1.00+0.00i>

Scalar Multiplication/Division c8*3, c8*3i, c8/3, c8/3i:
c8*3:  <1.1538+2.7692i>
c8*3i: <-2.7692+1.1538i>
c8/3:  <0.1282+0.3077i>
c8/3i: <0.3077+-0.1282i>

i's arbitary order
i's 5th order:   <0.00+1.00i>
i's 0.5th order: <0.71+0.71i>
i's 8th order:   <1.00+-0.00i>
i's -2th order:  <-1.00+-0.00i>
i's -2.5th order:<-0.71+0.71i>

Rotation from i
PI/4:       <-0.71+0.71i>
PI:         <-0.00+-1.00i>
1.5*PI:     <1.00+-0.00i>
2.3*PI:     <-0.81+0.59i>
270 degree: <1.00+-0.00i>
45 degree:  <-0.71+0.71i>
```

Expected Result:

Expected Result:



```
Show Functions for c6:
c6's abs:        3.6056
c6's angle:      0.5880
c6's angleDegree: 33.6901
c6's quadrant:   1

Complex Equality Check:
Is <1.123456789+1.123456789i> = <1.123456789+1.123456789i>? true
Is <1.123456789+1.123456789i> = <1.123456789+1.123456789i>? true
Is <1.123456789+1.123456789i> = <1.123456789+1.123456789i>? true
Is <1.123456789+1.123456789i> = <1.124567891+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.124567891+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.124567891+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.124567891i>? false
Is <1.123456789+1.123456789i> = <1.123456891+1.123456789i>? false
Is <1.123456789+1.123456789i> = <1.123456891+1.123456789i>? true
Is <1.123456789+1.123456789i> = <1.123456891+1.123456789i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.123456891i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.123456891i>? true
Is <1.123456789+1.123456789i> = <1.123456789+1.123456891i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.123456781i>? false
Is <1.123456789+1.123456789i> = <1.123456789+1.123456781i>? true
Is <1.123456789+1.123456789i> = <1.123456789+1.123456781i>? false
```

Reference Variables

LECTURE 7

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**.

Default Value for a Data Field

- 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);
  }
}
```

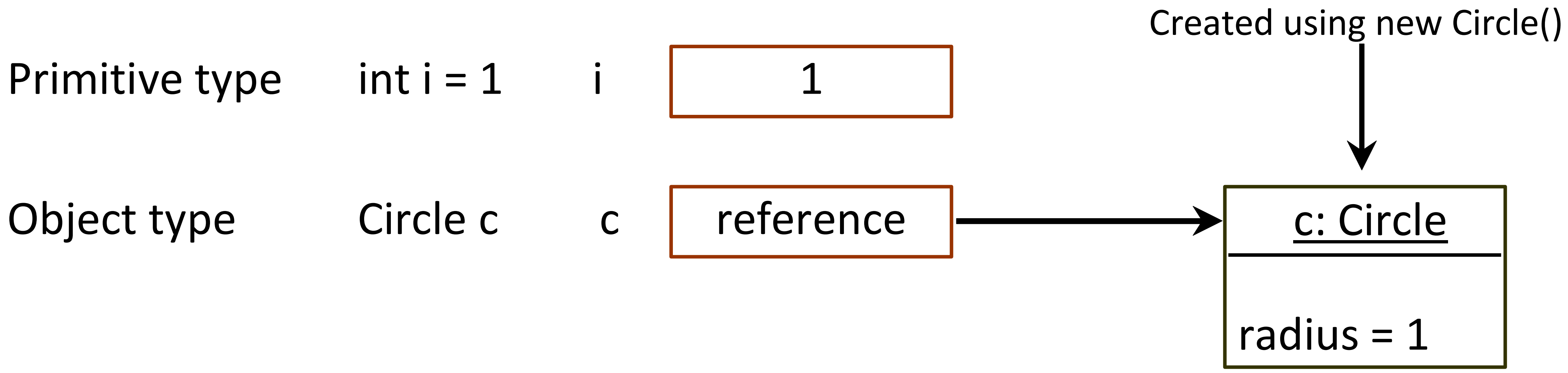
Example

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

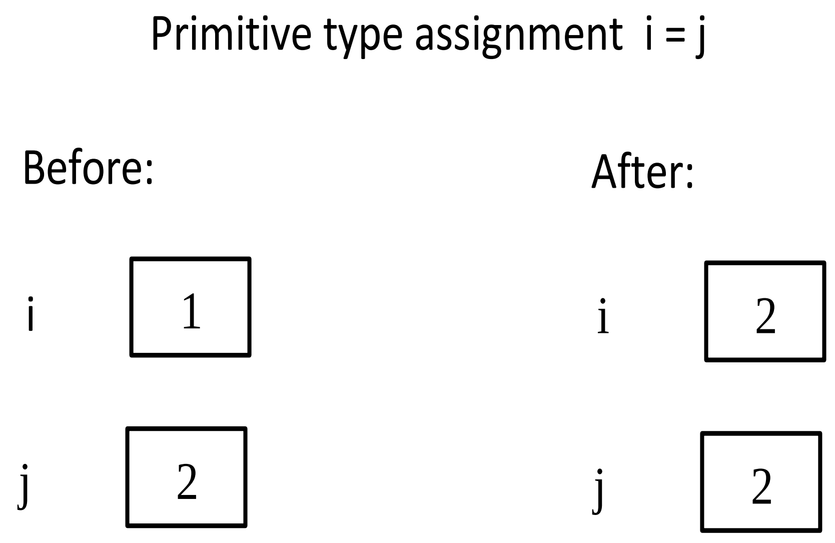
```
public class Test {
  public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
  }
}
```

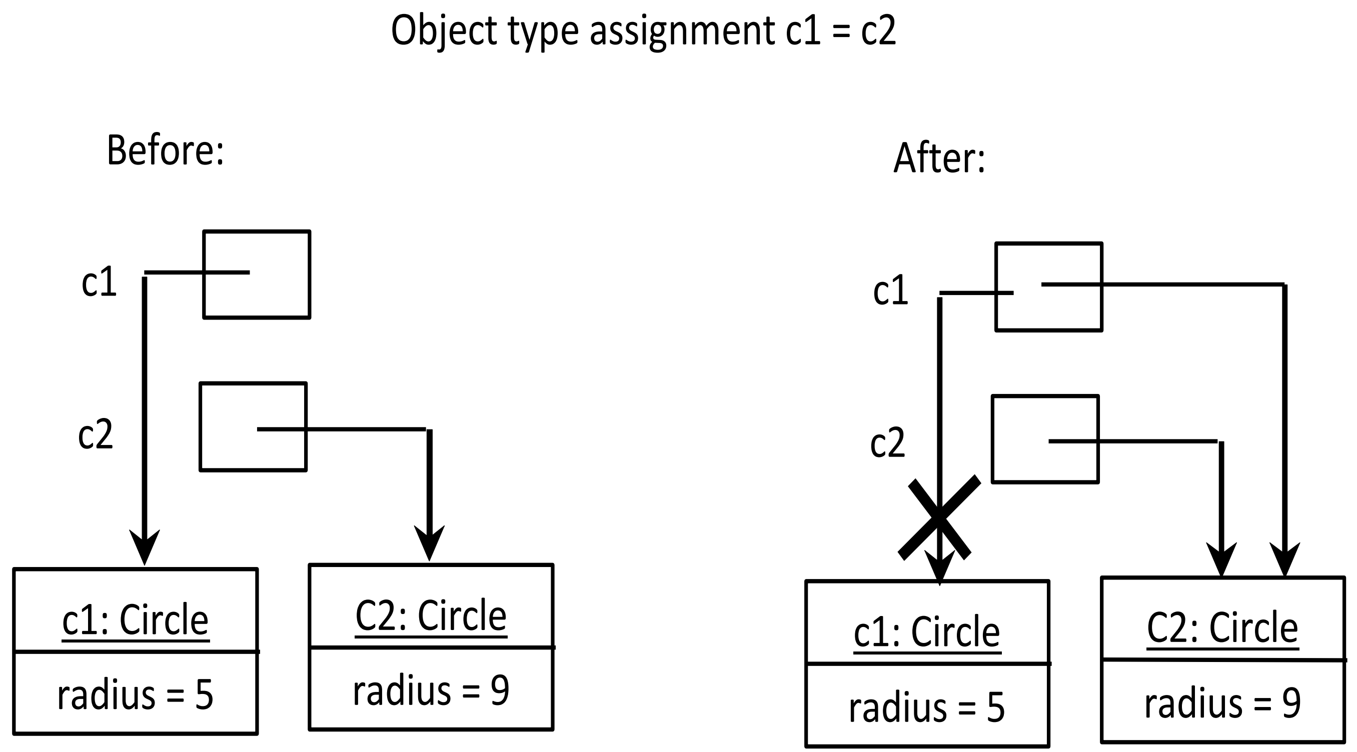
Compilation error: variables not initialized

Differences between Variables of Primitive Data Types and Object Types



Copying Variables of Primitive Data Types and Object Types





Garbage Collection

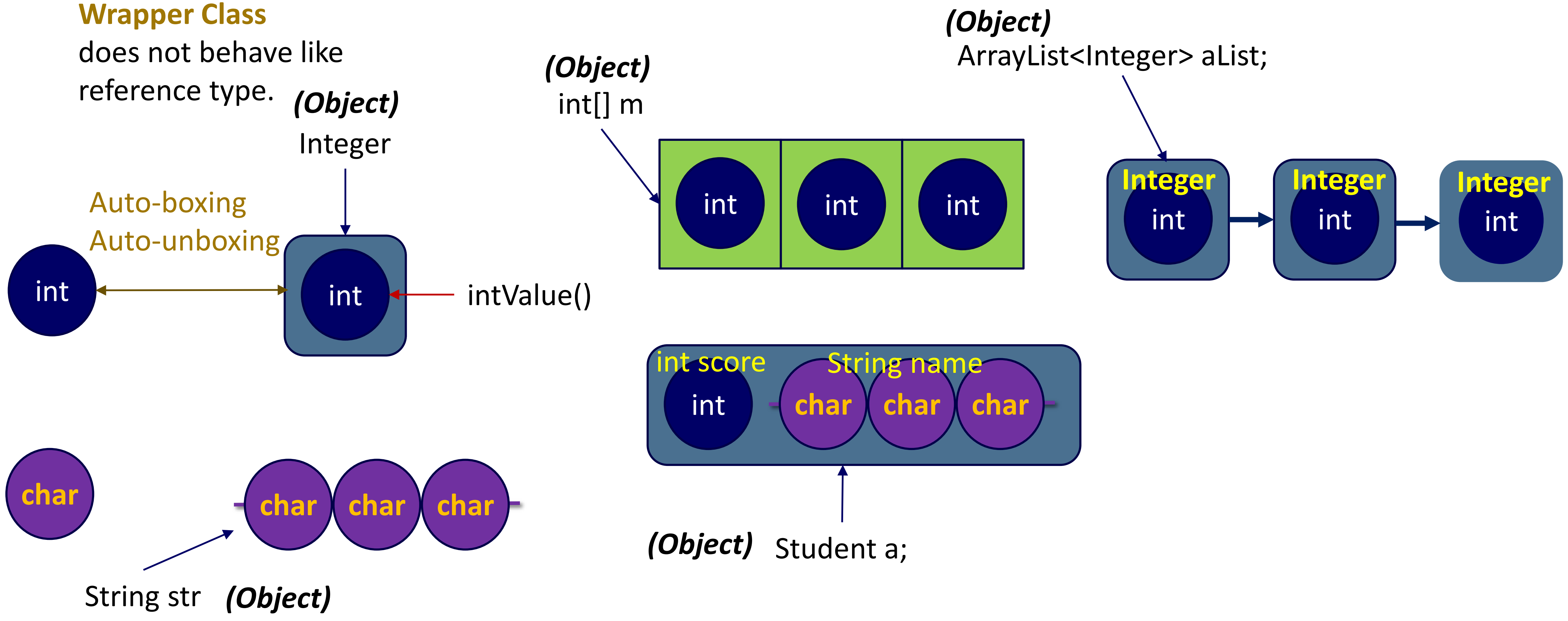
- As shown in the previous figure, after the assignment statement c1 = c2, c1 points to the same object referenced by c2. The object previously referenced by c1 is no longer referenced. This object is known as **garbage** (dangling object).
- Garbage is automatically collected by **JVM**.

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
int
int
intValue()
(Object)
int[] m
int
int
int
(Object)
ArrayList<Integer> aList;
Integer
int
Integer
int
Integer
int
int score
int
String name
char
char
char
(Object)
Student a;
char
char
char
char
String str
(Object)