
EE3206

Java Programming and Applications

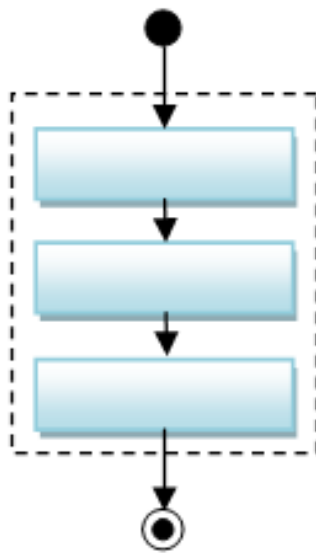
Lecture 2

Control Statement, Objects and Classes

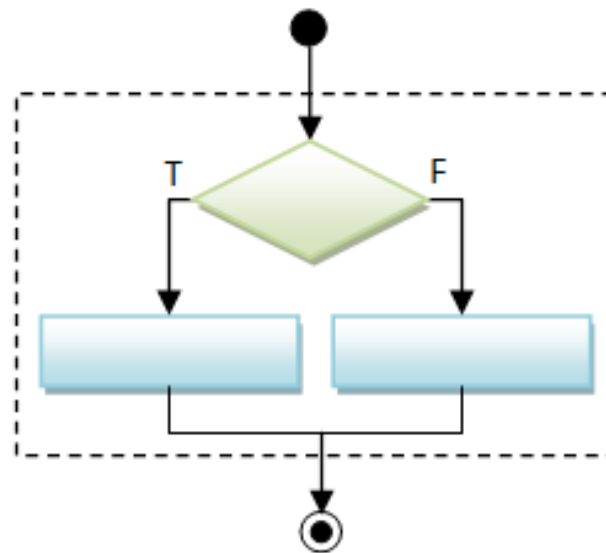
Intended Learning Outcomes

- ▶ Review control statement
- ▶ To understand relationship between objects and classes.
- ▶ To understand the use of constructors.
- ▶ To use private modifier to protect data fields
- ▶ To allow controlled access of private fields through getter or setter.
- ▶ To know the difference between instance and static variables and methods.
- ▶ To determine the scope of variables in the context of a class.
- ▶ To use the keyword *this* as the reference to the current running object
- ▶ To store and process objects in arrays.

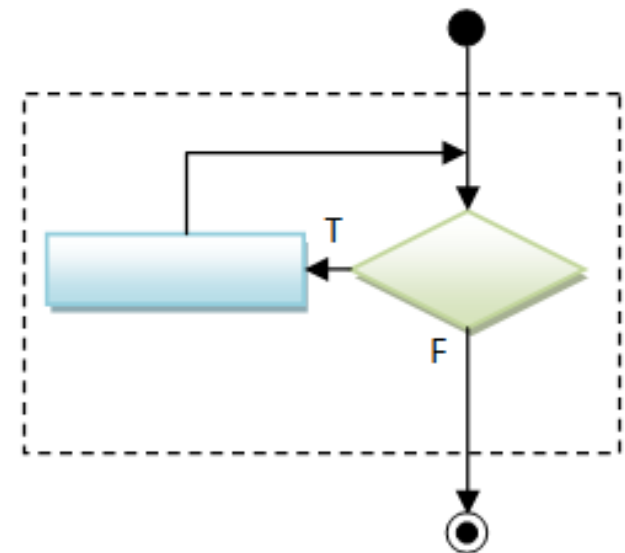
Flow of Control



Sequential



Conditional (Decision)



Loop (Iteration)

Sequential Flow Control

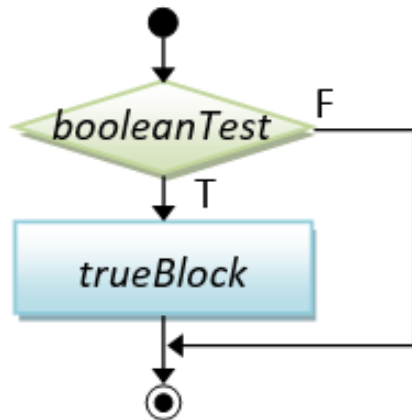
- ▶ A program is a sequence of instructions executing one after another in a predictable manner.
- ▶ Sequential flow is the most common and straight-forward, where programming statements are executed in the order that they are written - from top to bottom in a sequential manner.

Conditional Flow Control

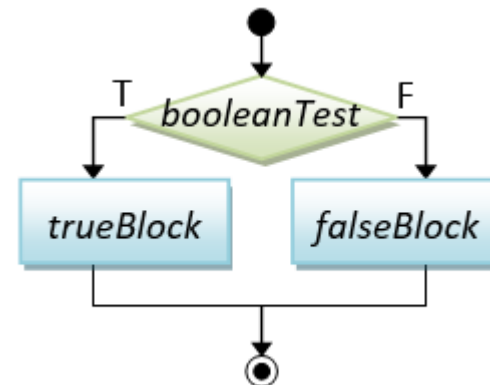
- ▶ There are a few types of conditionals,
 - ▶ if-then,
 - ▶ if-then-else,
 - ▶ nested-if,
 - ▶ switch-case-default,
 - ▶ and conditional expression.

if-then and if-then-else

```
// if-then
if (booleanTest) {
    trueBlock;
}
// next statement
```

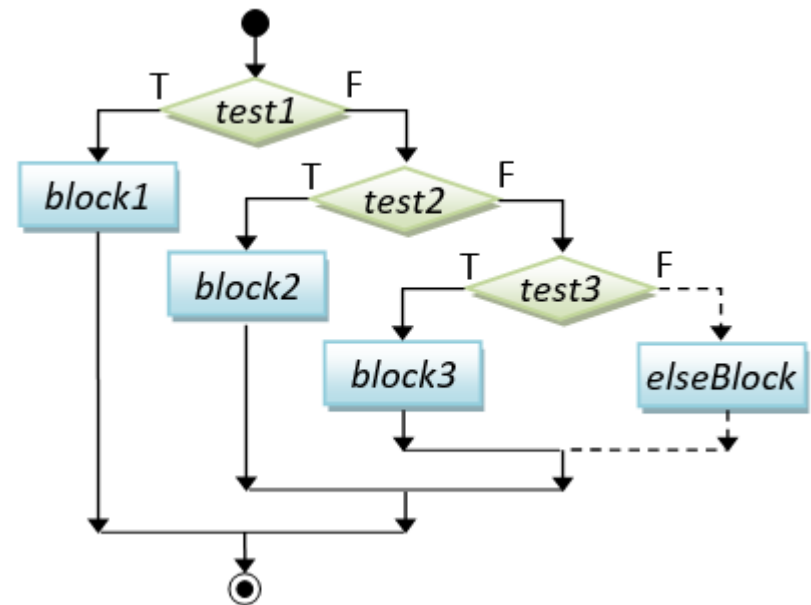


```
// if-then-else
if (booleanTest) {
    trueBlock;
} else {
    falseBlock;
}
// next statement
```



Nested-if

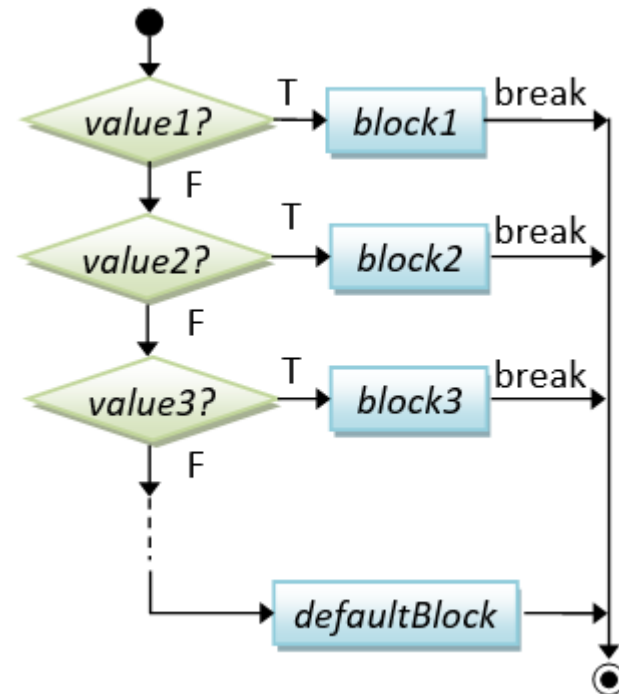
```
// nested-if
if (booleanTest1) {
    block1;
} else if (booleanTest2) {
    block2;
} else if (booleanTest3) {
    block3;
} else if (booleanTest4) {
    .....
} else {
    elseBlock;
}
// next statement
```



switch-case-default

```
// switch-case-default
switch (selector) {
  case value1:
    block1; break;
  case value2:
    block2; break;
  case value3:
    block3; break;
  .....
  case valueN:
    blockN; break;
  default: // not the above
    defaultBlock;
}
// next statement
```

- Primitive data types: byte, short, char, and int
- String class
- Enumerated types (enum)



Conditional Expression (... ? ... : ...)

► booleanExpr ? trueExpr : falseExpr

```
int num1 = 9, num2 = 8, max;
max = (num1 > num2) ? num1 : num2; // RHS returns num1 or num2
// same as
if (num1 > num2) {
    max = num1;
} else {
    max = num2;
}

int value = -9, absValue;
absValue = (value > 0) ? value : -value; // RHS returns value or -value
// same as
if (value > 0) absValue = value;
else absValue = -value;

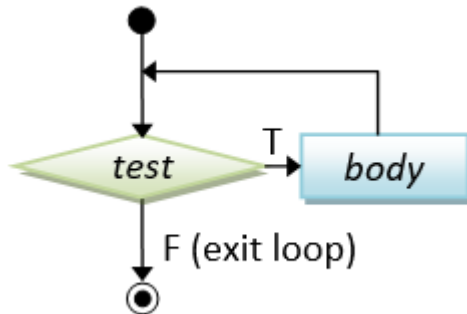
int mark = 48;
System.out.println((mark >= 50) ? "PASS" : "FAIL"); // Return "PASS" or "FAIL"
// same as
if (mark >= 50) System.out.println("PASS");
else System.out.println("FAIL");
```

Loop Flow Control

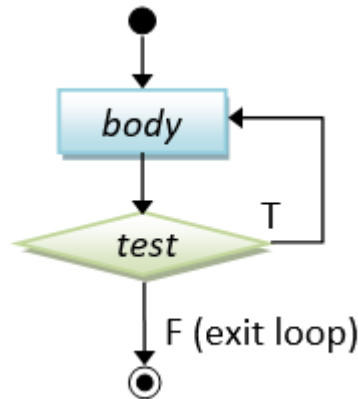
- ▶ There are a few types of loops:
 - ▶ *for*,
 - ▶ *while* (*while-do*),
 - ▶ and *do-while*.

Loop Flow Control

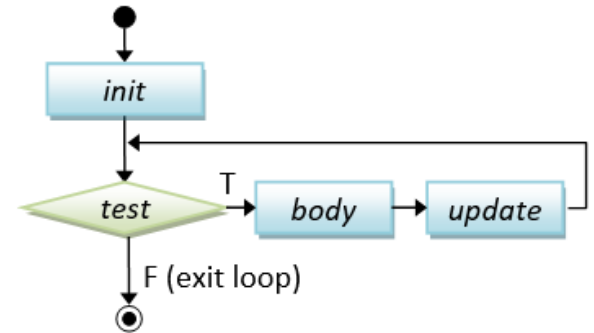
```
// while-do loop
while (booleanTest) {
    body;
}
// next statement
```



```
// do-while loop
do {
    body;
} while (booleanTest;
// next statement
// Need a semi-colon to
// terminate statement
```

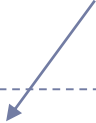


```
// for-loop
for (init; booleanTest; update) {
    body;
}
// next statement
```



Warm Up Exercise

- ▶ Before we continue, let's recall your memories...
- ▶ Problem:
 - ▶ In a 2D space, find the length of a line.
- ▶ What to do:
 1. Model the **Point** and **Line** objects in this problem domain
 2. Write the (business) logic to calculate the distance



```
public class WarmUp {  
    public static void main(String[] args) {  
        Point start = new Point(2, 2);  
        Point end = new Point(3, 3);  
        Line line = new Line(start, end);  
        System.out.println("The length is " + line.findLength());  
    }  
}
```

Modeling Point and Line

```
class Point {  
    int x;  
    int y;  
  
    Point(int n1, int n2) {  
        x = n1;  
        y = n2;  
    }  
}
```

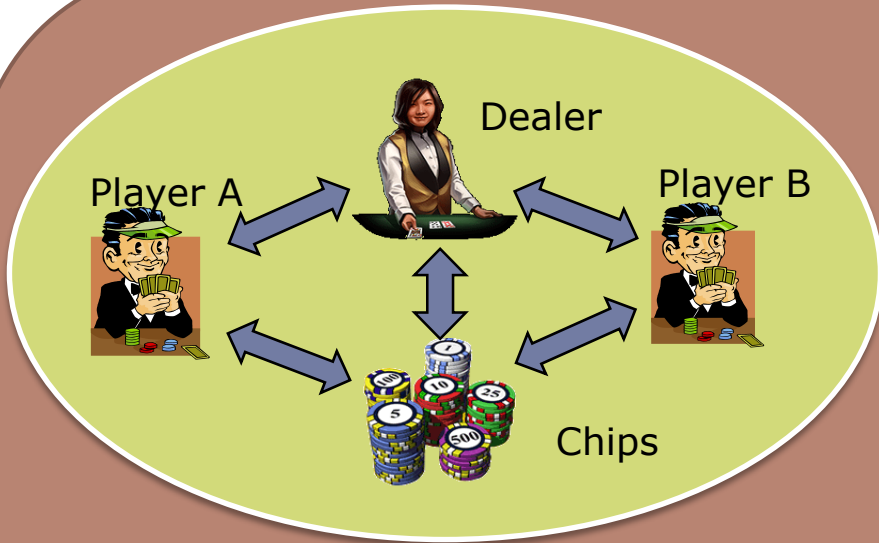
use

```
class Line {  
    Point start;  
    Point end;  
  
    Line(Point n1, Point n2) {  
        start = n1;  
        end = n2;  
    }  
  
    double findLength() {  
        return Math.sqrt(  
            Math.pow(start.x - end.x, 2) +  
            Math.pow(start.y - end.y, 2)  
        );  
    }  
}
```

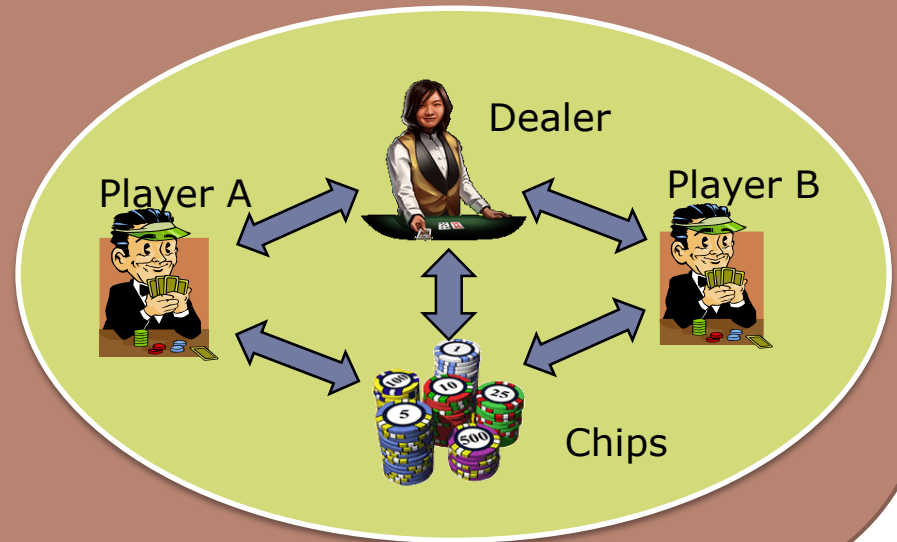
Raise of OOP

- ▶ Object-oriented programming (OOP) has roots that can be traced to the 1960s. **As hardware and software became increasingly complex, quality was often compromised.** Researchers studied ways to maintain software quality and developed object-oriented programming in part to address common problems by strongly emphasizing **discrete, reusable units of programming logic.**
- ▶ OOP involves programming using objects. An object represents an entity in the real world that can be distinctly identified.
 - ▶ For example, a student, a car, a circle, a button, and even a loan can all be viewed as objects.
- ▶ In OOP, each object is capable of receiving messages, processing data, and sending messages to other objects and can be viewed as an independent 'machine' with a distinct role or responsibility.
- ▶ Consider a casino game:
 - ▶ What else?
 - ▶ Table, Room, VipRoom, Account, Card, Dice, ...etc
 - ▶ Game types – Slots, Black Jack, Poker...etc





CASINO



Programming Paradigms

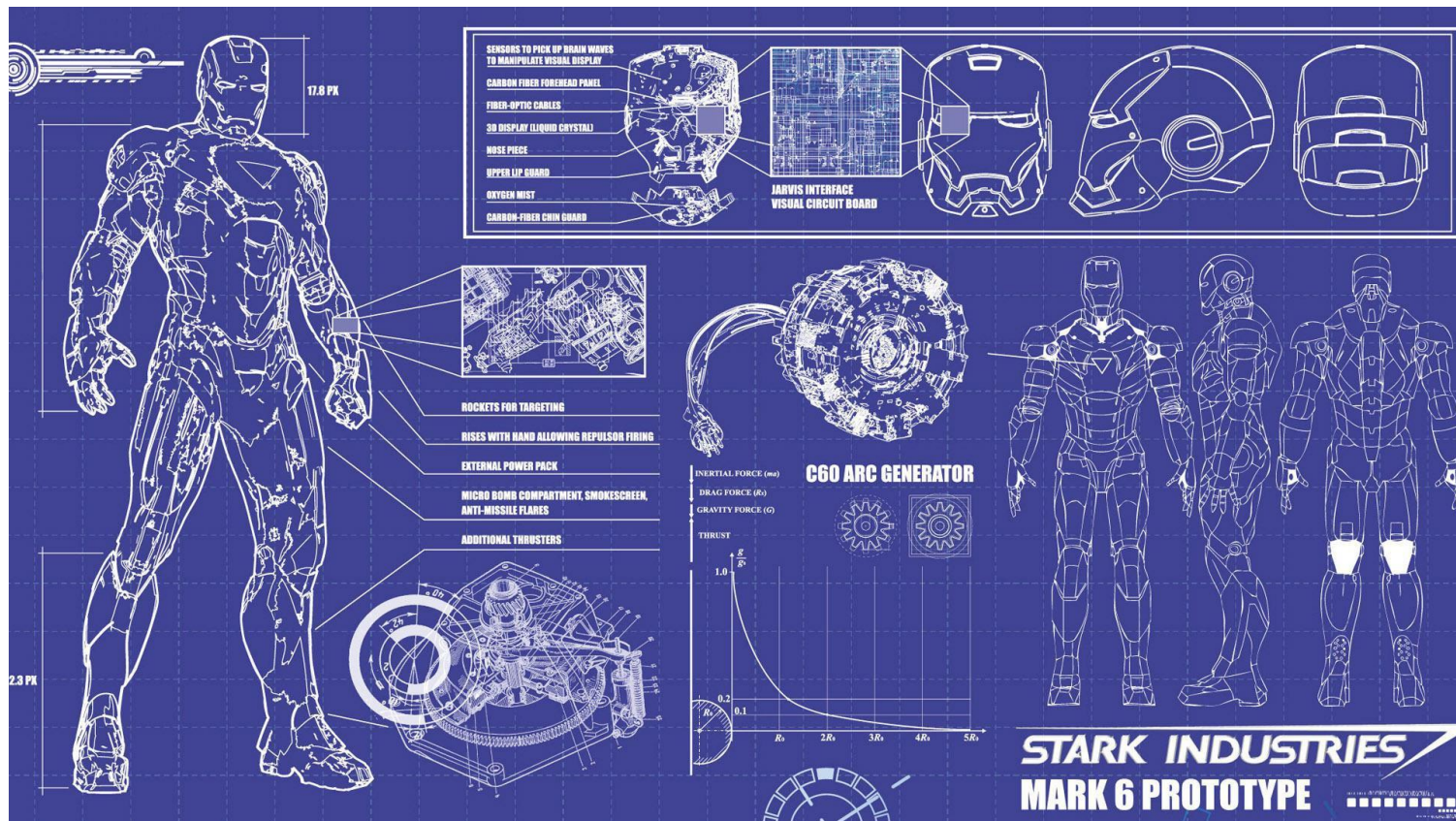
OOP	Procedural Programming
Programming task is broken down into objects that expose behavior (methods) and data (attributes)	Programming task is broken down into a collection of variables, data structures, and subroutines
Program is viewed as a collection of objects interacting with each other	Program simply contains a series of computational steps to be carried out
Self-contained module, an object operates on its own data structure	Uses procedures to operate on data structures

Objects

- ▶ An object has a unique **identity**, **state**, and **behaviors**.
 - ▶ The identity is the name of an object.
 - ▶ Student **peter** = new Student(); // object name
 - ▶ The state of an object consists of a set of data fields (also known as properties) with their current values.
 - ▶ peter.**age** // data field
 - ▶ peter.**sid**
 - ▶ peter.**gpa**
 - ▶ The behavior of an object is defined by a set of methods.
 - ▶ peter.**graduate()** // methods
 - ▶ peter.**promoteToNextYear()**

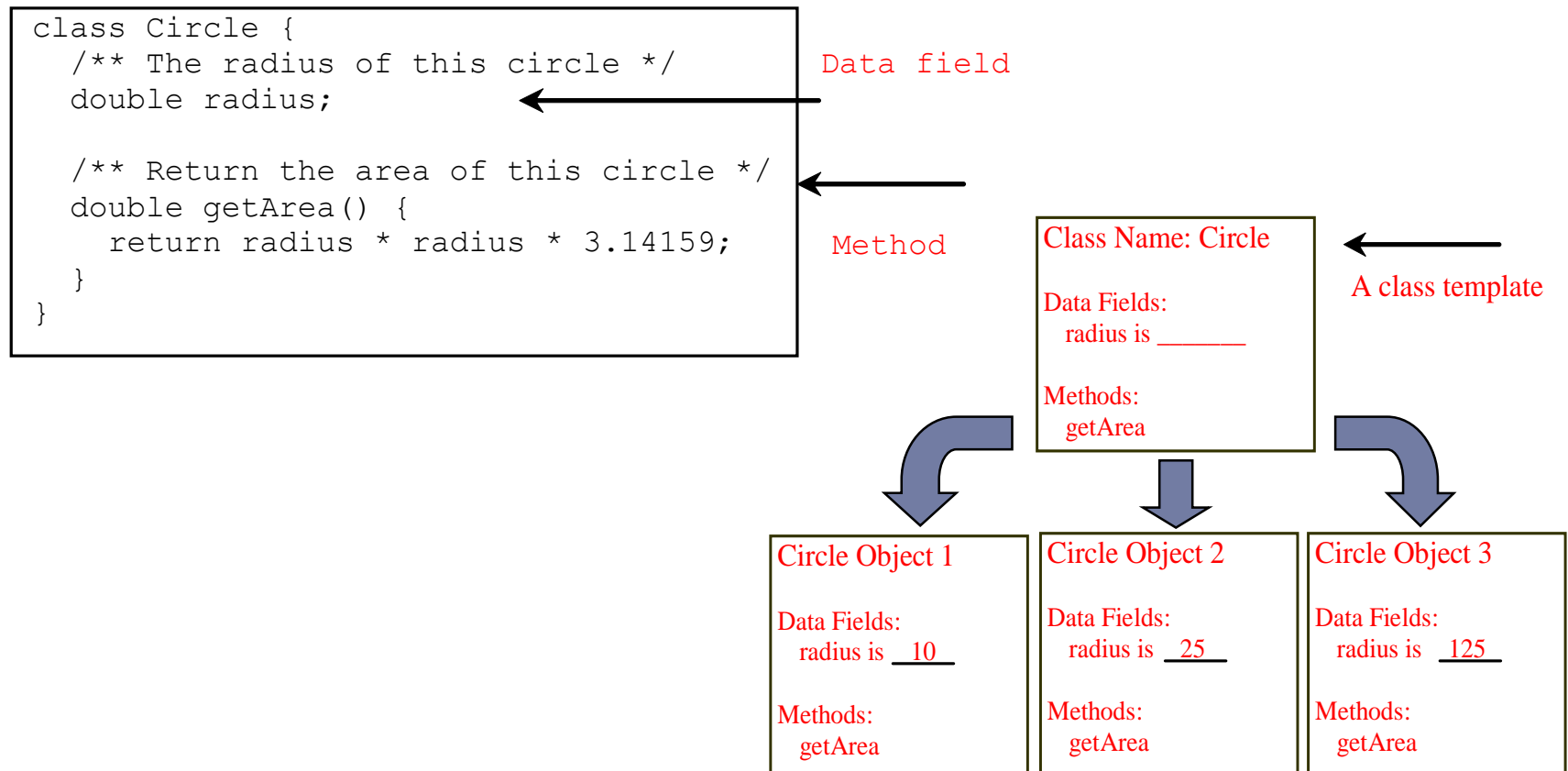
Relationship between Classes and Objects

- ▶ What about classes then?
 - ▶ Class is the blueprint for constructing objects



Classes

- ▶ Class is a construct that defines objects of the same type.
- ▶ A class tells what its objects possess.



Constructors

- ▶ A class provides a special type of methods, known as constructors, which are automatically invoked when an object is created.
- ▶ Constructors must have the same name as the class itself.
- ▶ Constructors do not have a return type — not even void.
- ▶ A constructor with no parameters is called **“no-arg” constructor**.

```
class Circle {  
    /** The radius of this circle */  
    double radius = 1.0;  
  
    /** Construct a circle object */  
    Circle() {  
        // no-arg constructor  
        // initialize data fields with default values  
    }  
  
    /** Construct a circle object */  
    Circle(double newRadius) {  
        radius = newRadius;  
    }  
  
    /** Return the area of this circle */  
    double getArea() {  
        return radius * radius * 3.14159;  
    }  
}
```

Using Constructors

- ▶ Constructors cannot be invoked directly as a normal method. They are invoked using the new operator when an object is created.

- ▶ new ClassName();

- ▶ new Student(); // correct

- ▶ Student.Student(); // wrong

- ▶ peter.Student(); // wrong

- ▶ Constructors play the role of initializing objects. You should place your code of initialization inside a constructor.

- ▶ new Circle(); // without args

- ▶ new Circle(5.0); // with args

Default Constructor

- ▶ A class may be declared without constructors. In this case, a **no-arg constructor with an empty body** is implicitly declared in the class.
 - ▶ Automatically inserted by compiler
- ▶ This constructor, called a default constructor, is provided automatically only if no constructors are explicitly declared in the class.

Declaring Objects

- ▶ Similar to declaring a variable of primitive data types, you can declare a reference variable for an object, using the syntax:
 - ▶ `ClassName objectRefVar;`
 - ▶ Example:
 - ▶ `Circle myCircle;` *// used to hold the reference value, similar to a C/C++ pointer*
- ▶ To reference an object, assign the object to a reference variable.
 - ▶ `ClassName objectRefVar = new ClassName();`
 - ▶ Example:
 - ▶ `Circle myCircle = new Circle();` *// RHS creates an object and return its reference value*
- ▶ If a reference type variable does not reference any object, the data field holds a special literal value, null.
 - ▶ `Circle myCircle;` *// declaration only, implicitly null*
 - ▶ `Circle myCircle = null;` *// equivalent*

Accessing Objects

- ▶ Referencing the object's data:
 - ▶ `objectRefVar.data`
 - ▶ Example:
 - ▶ `double myRadius = myCircle.radius;`
- ▶ Invoking the object's method:
 - ▶ `objectRefVar.methodName(arguments)`
 - ▶ Example:
 - ▶ `myCircle.getArea();`
- ▶ Objective:
 - ▶ Demonstrate creating objects, accessing data, and using methods.

TestCircle1

```
1 package ex2;
```

```
2 /**
3  *
4  */
5 public class TestCircle1 {
6
7     /** Main method */
8     public static void main(String[] args) {
9
10         // Create a circle with radius 5.0
11         Circle1 myCircle = new Circle1(5.0);
12         System.out.println("The area of the circle of radius " +
13             myCircle.radius + " is " + String.format("%.2f", myCircle.getArea()));
14
15         // Create a circle with radius 1
16         Circle1 yourCircle = new Circle1();
17         System.out.println("The area of the circle of radius " +
18             yourCircle.radius + " is " + String.format("%.2f", yourCircle.getArea()));
19
20         // Modify circle radius
21         yourCircle.radius = 100;
22         System.out.println("The area of the circle of radius " +
23             yourCircle.radius + " is " + String.format("%.2f", yourCircle.getArea()));
24     }
25 }
26
27
28
29
```

```
1 package ex2;
```

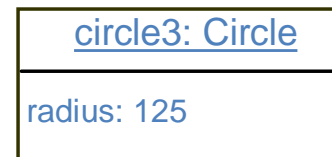
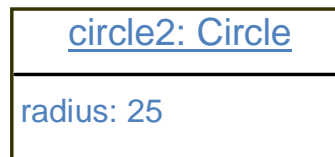
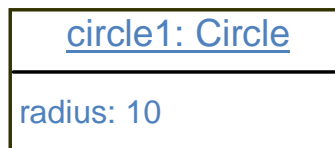
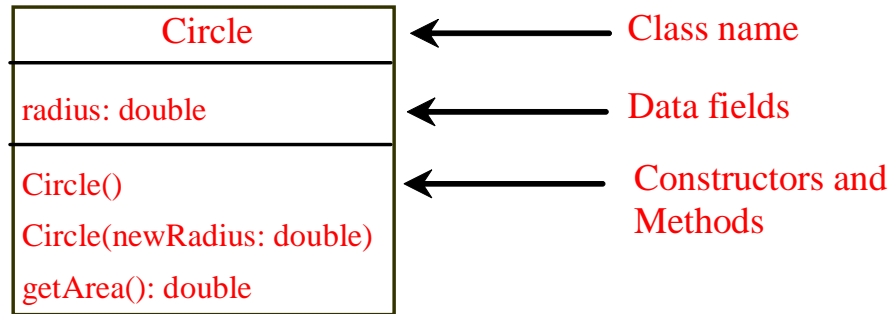
```
2
3 /// Define the circle class with two constructors
4 public class Circle1 {
5
6     double radius;
7
8     /** Construct a circle with radius 1 */
9     Circle1() {
10         radius = 1.0;
11     }
12
13     /** Construct a circle with a specified radius */
14     Circle1(double newRadius) {
15         radius = newRadius;
16     }
17
18     /** Return the area of this circle */
19     double getArea() {
20         return radius * radius * Math.PI;
21     }
22 }
23
```

```
The area of the circle of radius 5.0 is 78.54
The area of the circle of radius 1.0 is 3.14
The area of the circle of radius 100.0 is 31415.93
```

UML Class Diagram

- ▶ Unified Modeling Language (UML) is a standardized specification language for object modeling.
- ▶ It is a general-purpose modeling language that includes a graphical notation used to create an abstract model of a system, referred to as a *UML model*.

UML Class Diagram



← UML notation for objects

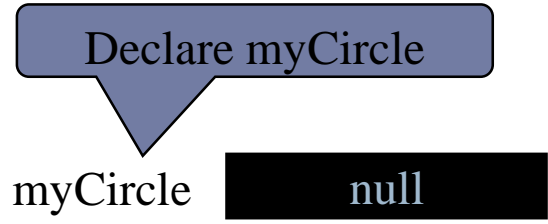
Initializing Objects

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

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

Declare myCircle



myCircle

null

Initializing Objects

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

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle null

Create a circle

: Circle

radius: 5.0

Initializing Objects

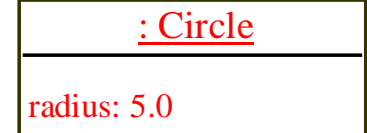
```
Circle myCircle = new Circle(5.0);
```

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle **reference value**

Assign object reference
to myCircle



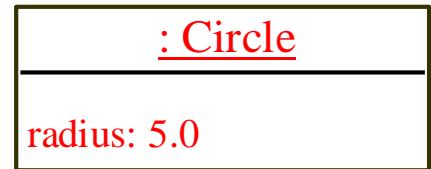
Initializing Objects

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

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle **reference value**



yourCircle **null**

Declare yourCircle

Initializing Objects

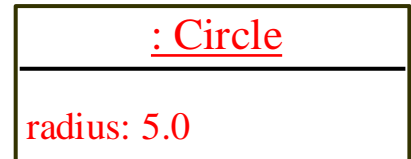
```
Circle myCircle = new Circle(5.0);
```

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle

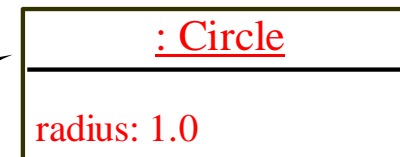
reference value



yourCircle

null

Create a new
Circle object



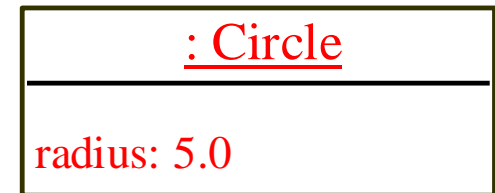
Initializing Objects

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

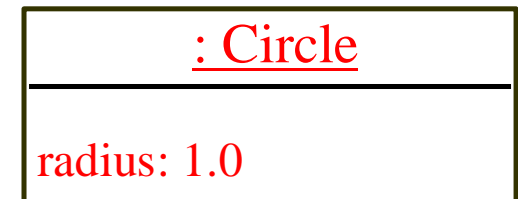
```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle **reference value**



yourCircle **reference value**



Assign object reference
to yourCircle

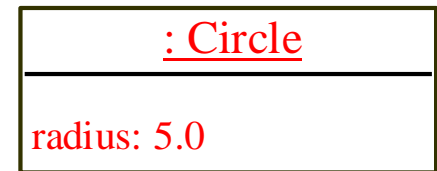
Initializing Objects

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

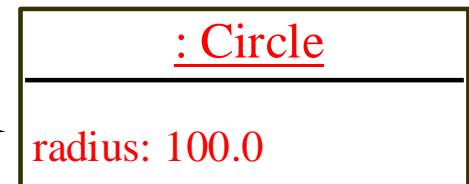
```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle **reference value**



yourCircle **reference value**



Change radius in
yourCircle

Data Fields

- ▶ Data fields refer to those variables declared in a class (whereas variables declared in methods are local variables)
 - ▶ The data fields can be of primitive types or reference types.
 - ▶ We have mentioned that String is a reference types.
 - ▶ For example, the following Student class contains mixed types of data field.

```
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'  
}
```

Default Value for a Data Field

- ▶ All data fields have a default value.
 - ▶ reference type = null
 - ▶ numeric type = 0
 - ▶ boolean type = false
 - ▶ char type = '\u0000' //16 bits unicode value in hex format

```
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);  
    }  
}
```

No Default Value for Local Variables

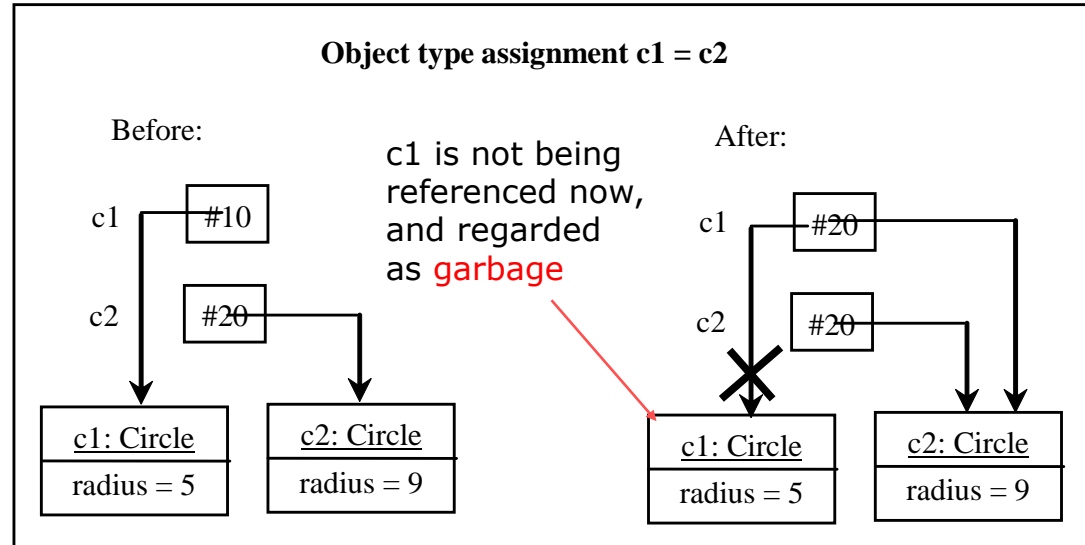
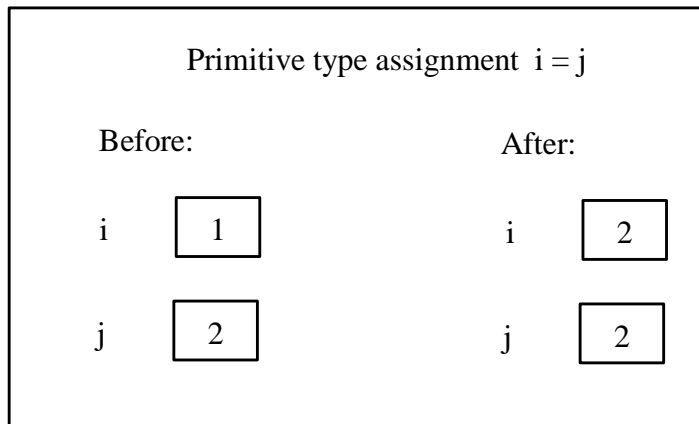
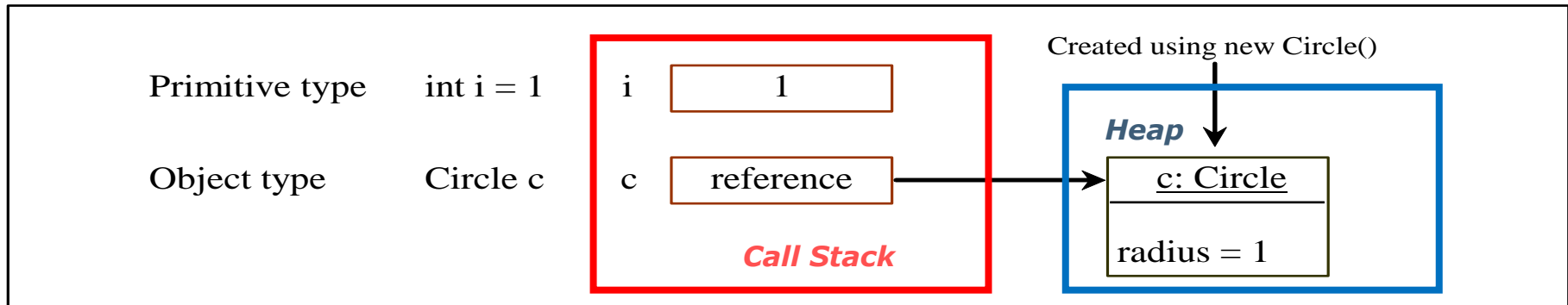
- ▶ However, Java assigns no default value to local variables inside method.

```
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

Difference between Primitive Type and Object Type



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.
 - ▶ Garbage is automatically collected by JVM.
 - ▶ You don't need to acquire/release memory by yourself.
- ▶ 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.

- ▶ For example:

```
Circle c = new Circle();
```

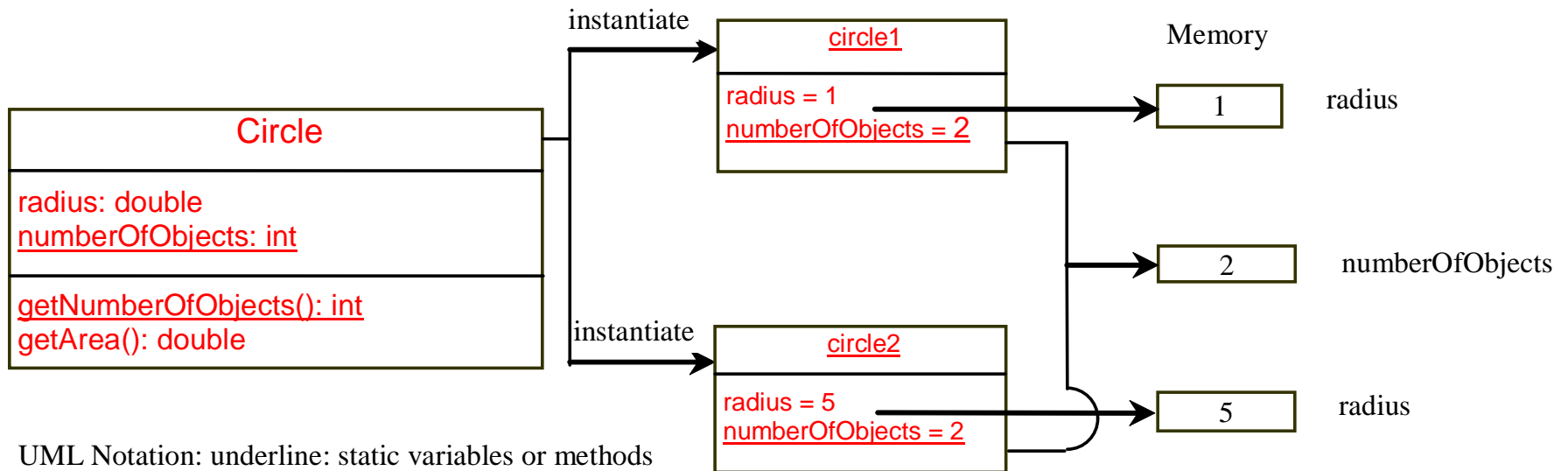
```
...
```

```
...           // after some operations and no need c anymore
```

```
c = null;
```

Static Modifier – Share Property

- ▶ Static modifier can be used in variables and methods.
- ▶ **They are not copied to each instances but registered in the class only.**
- ▶ Static variable/method can be accessed through a class reference.
 - ▶ E.g. `ClassName.varName` / `ClassName.methodName()`
- ▶ Static variables are shared by all the instances of the same class.
 - ▶ Each object has it's own radius (instance variable), but both circle1 and circle2 share the same numberOfObjects (class variable).



UML Notation: underline: static variables or methods

TestCircle2

```
package ex2;

public class TestCircle2 {

    /** Main method */
    public static void main(String[] args) {
        // Create c1
        Circle2 c1 = new Circle2();

        // Display c1 BEFORE c2 is created
        System.out.println("Before creating c2");
        System.out.println("c1 is : radius (" + c1.radius +
            ") and number of Circle objects (" + c1.numberOfObjects + ")");

        // Create c2
        Circle2 c2 = new Circle2(5);

        // Change the radius in c1
        c1.radius = 9;

        // Display c1 and c2 AFTER c2 was created
        System.out.println("\nAfter creating c2 and modifying c1's radius to 9");
        System.out.println("c1 is : radius (" + c1.radius +
            ") and number of Circle objects (" + c1.numberOfObjects + ")");
        System.out.println("c2 is : radius (" + c2.radius +
            ") and number of Circle objects (" + c2.numberOfObjects + ")");
    }
}
```

Before creating c2

c1 is : radius (1.0) and number of Circle objects (1)

After creating c2 and modifying c1's radius to 9

c1 is : radius (9.0) and number of Circle objects (2)

c2 is : radius (5.0) and number of Circle objects (2)

```
package ex2;

public class Circle2 {

    /** The radius of the circle */
    double radius;
    /** The number of the objects created */
    static int numberOfObjects = 0;

    /** Construct a circle with radius 1 */
    Circle2() {
        radius = 1.0;
        numberOfObjects++;
    }

    /** Construct a circle with a specified radius */
    Circle2(double newRadius) {
        radius = newRadius;
        numberOfObjects++;
    }

    /** Return numberOfObjects */
    static int getNumberOfObjects() {
        return numberOfObjects;
    }

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


Visibility Modifiers

- ▶ Package level (no visibility modifier)
 - ▶ By default, the class, variable or method can be accessed by any class in the same package.
- ▶ Public level (modifier: **public**)
 - ▶ No restriction of access
 - ▶ The class, variable or method is visible to any class in any package.
- ▶ Private level (modifier: **private**)
 - ▶ The variable or method can be accessed only by the declaring class itself.

Restricting Accessibility

► Example I

package p1;

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

```
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();  
    }  
}
```

► Ex2

package p1;

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

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

package p2;

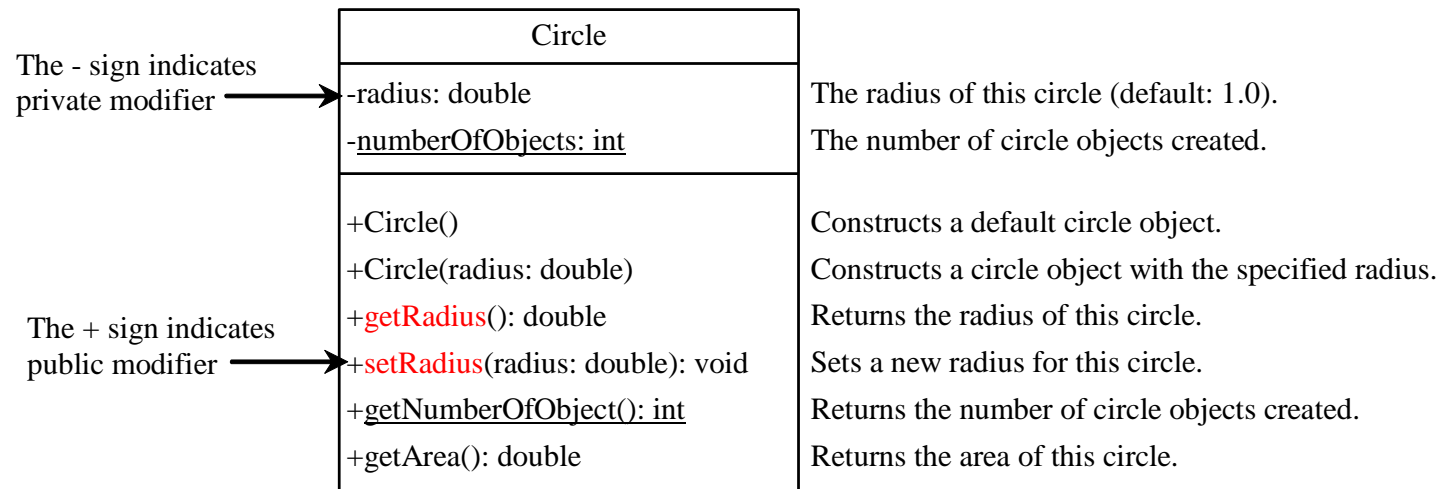
```
public class C3 {  
    cannot access C1;  
    can access C2;  
}
```

Problem of Public Properties

- ▶ I have a bank account object, says myAccount
- ▶ Users can read my balance by
 - ▶ `myAccount.balance`
- ▶ They can also change it by
 - ▶ `myAccount.balance = 999999`
- ▶ Scenario 1: I only want to let users read my balance. I don't want them to change it!
- ▶ Scenario 2: I allow users to update my balance only if they are authorized to do so.

Data Encapsulation

- ▶ **Data encapsulation**, also known as data hiding, is the mechanism whereby the implementation details of a class are kept hidden from the user. **The user can only perform a restricted set of operations on the hidden (private) members of the class by executing special (public) methods.** As a result:
 - ▶ Data fields are being protected from illegal access
 - ▶ Maintenance of a class becomes easier in the long run
- ▶ We can then allow controlled access through public methods of the same class. These methods are called getter and setter (aka accessor and mutator). They are used to read and modify private properties.



TestCircle3

```
package ex2;

// TestCircle3.java: Demonstrate private modifier
public class TestCircle3 {

    /** Main method */
    public static void main(String[] args) {
        // Create a Circle with radius 5.0
        Circle3 myCircle = new Circle3(5.0);
        System.out.println("The area of the circle of radius " +
            myCircle.getRadius() + " is " + String.format("%.2f", myCircle.getArea()));

        // Increase myCircle's radius by 10%
        myCircle.setRadius(myCircle.getRadius() * 1.1);
        System.out.println("The area of the circle of radius " +
            myCircle.getRadius() + " is " + String.format("%.2f", myCircle.getArea()));
    }
}
```

```
--- EXECUTING (default-CL) @ 202006 ---
The area of the circle of radius 5.0 is 78.54
The area of the circle of radius 5.5 is 95.03
-----
```

```
package ex2;

public class Circle3 {

    /** The radius of the circle */
    private double radius = 1;
    /** The number of the objects created */
    private static int numberOfObjects = 0;

    /** Construct a circle with radius 1 */
    public Circle3() {
        numberOfObjects++;
    }

    /** Construct a circle with a specified radius */
    public Circle3(double newRadius) {
        radius = newRadius;
        numberOfObjects++;
    }

    /** Return radius */
    public double getRadius() {
        return radius;
    }

    /** Set a new radius */
    public void setRadius(double newRadius) {
        radius = (newRadius >= 0) ? newRadius : 0;
    }

    /** Return numberOfObjects */
    public static int getNumberOfObjects() {
        return numberOfObjects;
    }

    /** Return the area of this circle */
    public double getArea() {
        return radius * radius * Math.PI;
    }
}
```

Immutable Objects and Classes

- ▶ If the contents of an object cannot be changed once the object is created, the object is called an immutable object and its class is called an immutable class.
- ▶ If you delete the **setRadius** method in the preceding example, the Circle class would be immutable because radius is private and cannot be changed without a set method.

Student is mutable or immutable?



```
public class Student {  
    private int id;  
    private BirthDate birthDate;  
  
    public Student(int ssn,  
        int year, int month, int day) {  
        id = ssn;  
        birthDate = new BirthDate(year, month, day);  
    }  
  
    public int getId() {  
        return id;  
    }  
  
    public BirthDate getBirthDate() {  
        return birthDate;  
    }  
}
```

```
public class BirthDate {  
    private int year;  
    private int month;  
    private int day;  
  
    public BirthDate(int newYear,  
        int newMonth, int newDay) {  
        year = newYear;  
        month = newMonth;  
        day = newDay;  
    }  
  
    public void setYear(int newYear) {  
        year = newYear;  
    }  
}
```

What make Class Immutable?

- ▶ A class with all private data fields and without mutators is not necessarily immutable.
- ▶ For a class to be immutable, it must:
 - ▶ mark all data fields private;
 - ▶ provide no setter (mutator) methods;
 - ▶ ***provide no getter (accessor) methods that would return a reference to a mutable data field object.*

Scope of Variables in Class

▶ Data fields

- ▶ The scope of data fields is the entire class. They can be declared anywhere inside a class, though this is not preferred (lower readability).
- ▶ They are implicitly initialized with default value.

▶ Local variables

- ▶ The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable.
- ▶ A local variable must be initialized explicitly before it can be used.

The Keyword - **this**

- ▶ The keyword **this** is a reference to **the current object context (i.e. the executing object)**.
- ▶ It is automatically available in any object context.
- ▶ Two common usages:
 - ▶ To explicitly refer to an instance's data field.
 - ▶ To invoke an overloaded constructor of the same class.

```
public class Circle {  
    private double radius;
```

```
    public Circle(double radius) {  
        this.radius = radius;  
    }
```

→ this must be explicitly used to reference the data field radius of the object being constructed

```
    public Circle() {  
        this(1.0);  
    }
```

→ this is used to invoke another constructor

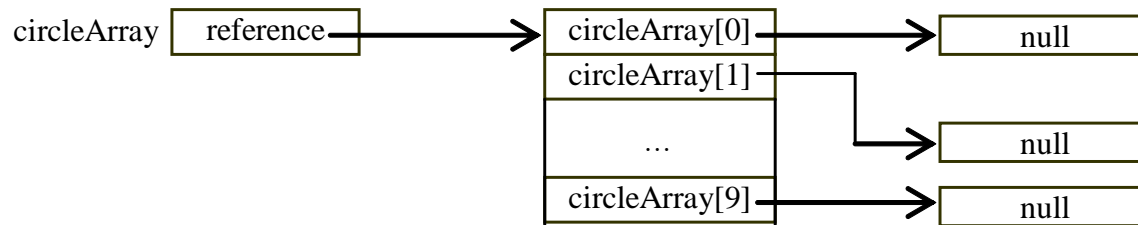
```
    public double getArea() {  
        return this.radius * radius * Math.PI;  
    }
```

↓ ↓
Every instance variable belongs to an instance represented by this, which is normally omitted

Array of Objects

- ▶ An array of objects is actually an array of reference variables.

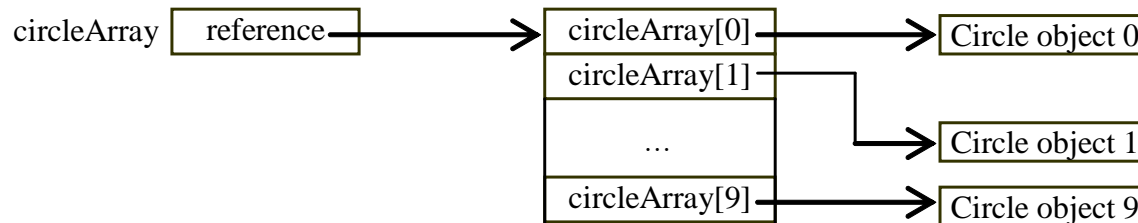
- ▶ `Circle[] circleArray = new Circle[10];` *// no circles !!!*



- ▶ The above code only creates an Array object but not Circle objects. For each elements, you need to create them one by one:

```
for(int i=0; i<circleArray.length; i++)
```

```
    circleArray[i] = new Circle();
```



Code sample

```
/** Create an array of Circle objects */
public static Circle3[] createCircleArray() {
    Circle3[] circleArray = new Circle3[5];

    for (int i = 0; i < circleArray.length; i++) {
        circleArray[i] = new Circle3(10 + Math.random() * 90);
    }

    // Return Circle array
    return circleArray;
}
```

Introduction to Useful Classes

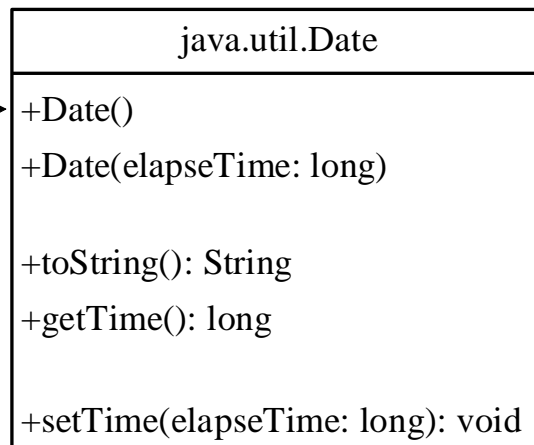
Date

Random

The Date Class

- ▶ You can use the `java.util.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. For example,
 - ▶ `java.util.Date date = new java.util.Date();`
 - ▶ `System.out.println(date.toString());`
- ▶ The above code will output a string representation of the date like:
 - ▶ Wed Sep 30 14:35:46 CST 2009

The + sign indicates
public modifier



- 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 Random Class

- ▶ You have used `Math.random()` to obtain a random double value between 0.0 and 1.0 (excluding 1.0). A more useful random number generator is provided in the `java.util.Random` class.

java.util.Random	
+Random()	Constructs a Random object with the current time as its seed.
+Random(seed: long)	Constructs a Random object with a specified seed.
+nextInt(): int	Returns a random int value.
+nextInt(n: int): int	Returns a random int value between 0 and n (exclusive).
+nextLong(): long	Returns a random long value.
+nextDouble(): double	Returns a random double value between 0.0 and 1.0 (exclusive).
+nextFloat(): float	Returns a random float value between 0.0F and 1.0F (exclusive).
+nextBoolean(): boolean	Returns a random boolean value.

Random Class Example

- ▶ If two Random objects have the same seed, they will generate identical sequences of numbers. For example, the following code creates two Random objects with the same seed 3.

```
Random random1 = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++)
    System.out.print(random1.nextInt(1000) + " ");
Random random2 = new Random(3);
System.out.print("\nFrom random2: ");
for (int i = 0; i < 10; i++)
    System.out.print(random2.nextInt(1000) + " ");
```



```
From random1: 734 660 210 581 128 202 549 564 459 961
From random2: 734 660 210 581 128 202 549 564 459 961
```


In-class practice

- ▶ Write a Java program to find the longest words in a dictionary.

Example-1:

```
{"cat","flag","green","country","javaee3206"}
```

Result: " javaee3206 "



Example-2:

```
{"cat","dog","red","is","am"}
```

Result: "cat", "dog", "red"

Some helpful method

- clear() / Clear the list
- Add() / add to the list

//search document for more details