

JAVATM

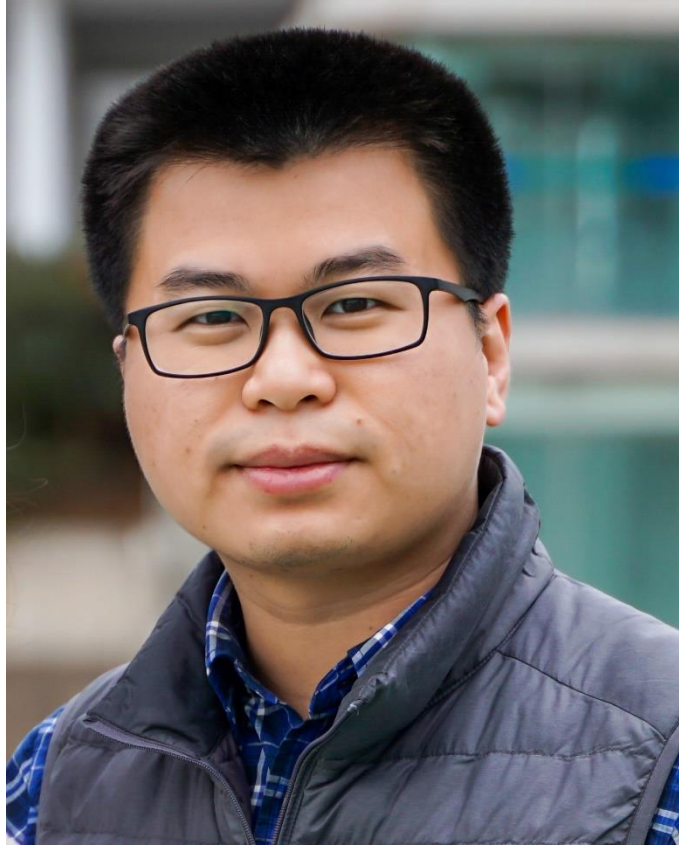


ĐỐI TƯỢNG & LỚP (CLASS)

GIẢNG VIÊN

TS. Hà Ngọc Long

THÔNG TIN GIẢNG VIÊN



TS. Hà Ngọc Long

Email: hnlng@hueuni.edu.vn

- Thông tin Giảng viên
 - Tiến sĩ ngành Hệ thống Thông tin và Truyền thông, Hàn Quốc.
 - Thạc sĩ ngành Kỹ thuật Quản trị Công nghiệp, Hàn Quốc.
 - Cử nhân ngành Hệ thống Thông tin Quản lý, Đại học Kinh tế Huế
- Hướng nghiên cứu:
 - Quản trị quy trình nghiệp vụ (Business Process Management).
 - Chuyển đổi số.
 - FinTech, RegTech
 - Tiền điện tử, hợp đồng thông minh, và Blockchain.
 - Khai phá quy trình nghiệp vụ (Process Mining)

MỤC TIÊU CỦA HỌC PHẦN

“Giới thiệu về ngôn ngữ lập trình hướng đối tượng Java. Tập trung giới thiệu cho sinh viên kỹ năng, kỹ thuật lập trình theo phương pháp hướng đối tượng”

TIÊU CHUẨN ĐÁNH GIÁ SINH VIÊN

Hình thức Đánh giá	% Điểm
Chuyên cần	5%
Phát Biểu & Thảo Luận	5%
Bài Tập Về Nhà	5%
Thực Hành & Thực Tế	15%
Kiểm tra giữa kỳ	15%
Bài tập cá nhân	10%
<i>Thi kết thúc học phần</i>	<i>45%</i>
Tổng cộng:	100%

NỘI DUNG MÔN HỌC

1

- Java cơ bản

2

- Nhập môn lập trình HĐT

3

- Lập trình HĐT với Java

4

- Lập trình đồ họa và sự kiện trong Java

5

- Cấu trúc dữ liệu nâng cao trong Java

TÀI LIỆU HỌC TẬP

- *Y. Daniel Liang* (2019) Introduction to Java Programming and Data Structures - Comprehensive Version (12th edition), *Pearson*
- *C Thomas Wu.* (2004). An introduction to object oriented programming with Java. *McGraw-Hill – Boston.*

Tài liệu hướng dẫn và phần mềm thực hành:

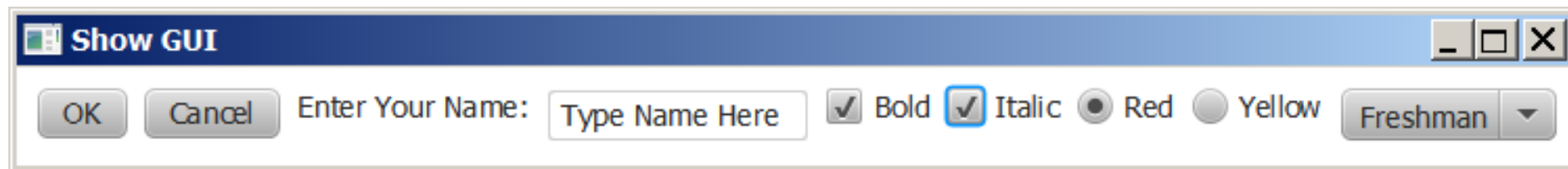
- Apache Netbeans IDE 12 (or higher)
- Java SE 17 (or higher)
- Videos bài giảng và hướng dẫn thực hành



Chapter 9 Objects and Classes

Motivations

- Sau khi học các chương trước, chúng ta đã có những kỹ năng giải quyết nhiều vấn đề lập trình bằng cách sử dụng câu lệnh *điều kiện*, *vòng lặp*, *phương thức* và *mảng*.
- Tuy nhiên, các tính năng Java này *không đủ* để chúng ta có thể phát triển giao diện người dùng đồ họa và hệ thống phần mềm hoàn chỉnh.
- Giả sử bạn muốn phát triển một giao diện người dùng đồ họa như hình dưới đây. Làm thế nào để ta có thể tạo ra nó?



Objectives (1/2)

- Mô tả các đối tượng và lớp, và sử dụng các lớp để mô hình hóa các đối tượng (§9.2).
- Sử dụng ký hiệu đồ họa **UML** để mô tả các lớp và đối tượng (§9.2).
- Trình bày cách xác định các lớp và tạo các đối tượng (§9.3).
- Tạo các đối tượng bằng cách sử dụng hàm tạo (§9.4).
- Truy cập các đối tượng thông qua các biến tham chiếu đối tượng (§9.5).
- Xác định một biến tham chiếu bằng cách sử dụng kiểu tham chiếu (§9.5.1).
- Truy cập dữ liệu và phương thức của một đối tượng bằng toán tử truy cập thành viên đối tượng (.) (§9.5.2).
- Xác định các trường dữ liệu của loại tham chiếu và gán giá trị mặc định cho các trường dữ liệu của đối tượng (§9.5.3).
- Phân biệt giữa biến tham chiếu đối tượng và biến kiểu dữ liệu nguyên thủy (§9.5.4).

Objectives (2/2)

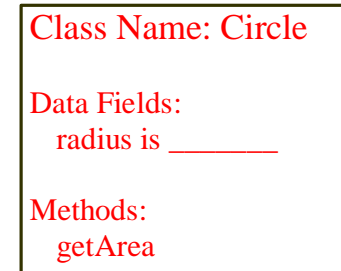
- Sử dụng các lớp thư viện Java **Date**, **Random** và **Point2D** (§9.6).
- Phân biệt giữa các biến và phương thức **instance** và **static** (§9.7).
- Xác định các trường dữ liệu riêng tư với các phương thức **get** và **set** thích hợp (§9.8).
- Đóng gói các trường dữ liệu để làm cho các lớp dễ bảo trì (§9.9).
- Phát triển các phương thức với đối số đối tượng và phân biệt giữa đối số kiểu nguyên thủy và đối số kiểu đối tượng (§9.10).
- Lưu trữ và xử lý các đối tượng trong mảng (§9.11).
- Tạo các **đối tượng bất biến** (immutable objects) từ các lớp bất biến (immutable classes) để bảo vệ nội dung của các đối tượng (§9.12).
- Xác định phạm vi của các biến trong ngữ cảnh của một lớp (§9.13).
- Sử dụng từ khóa **this** để tham chiếu đến chính đối tượng đang gọi (§9.14).

OO Programming Concepts

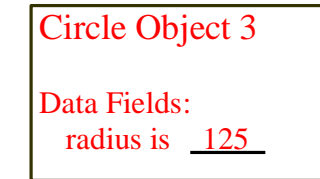
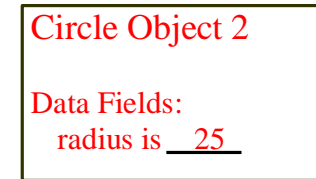
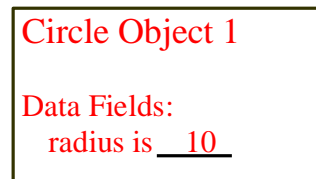
- *Lập trình hướng đối tượng* (OOP) liên quan đến việc lập trình sử dụng các *đối tượng*. Một đối tượng đại diện cho *một thực thể* trong thế giới thực có thể được xác định rõ ràng.
- **Ví dụ:** một học sinh, một cái bàn, một vòng tròn, một cái nút, và thậm chí một khoản vay nợ đều có thể được xem như là đồ vật.
- Một đối tượng có một danh tính, trạng thái và hành vi duy nhất.
- Trạng thái của một đối tượng bao gồm một tập hợp các trường dữ liệu (còn được gọi là thuộc tính - *properties*) với các giá trị hiện tại của chúng.
- Hành vi của một đối tượng được xác định bởi một tập hợp các *phương thức*.

Objects

- Một đối tượng có cả *trạng thái* và *hành vi*. **Trạng thái** định hình đối tượng và **hành vi** định hình cái mà đối tượng sẽ làm.



← A class template



← Three objects of the Circle class

- Một đối tượng (object) bao gồm 2 thông tin: **thuộc tính** và **phương thức**.
 - **Thuộc tính** là những thông tin, đặc điểm của đối tượng. **Ví dụ:** con người có các đặc tính như mắt, mũi, tay, chân...
 - **Phương thức** là những thao tác, hành động mà đối tượng đó có thể thực hiện. **Ví dụ:** một người sẽ có thể thực hiện hành động nói, đi, ăn, uống, ...

Classes (1/3)

- Một lớp là một (cấu trúc) *kiểu dữ liệu* bao gồm các *thuộc tính* và các *phương thức* được định nghĩa từ trước.
- Một lớp Java sử dụng các biến để xác định các trường dữ liệu và các phương thức để xác định các hành vi.
- Ngoài ra, một lớp cung cấp một loại phương thức đặc biệt, được gọi là phương thức tạo (constructor), được gọi để xây dựng các đối tượng từ một lớp.

Classes (2/3)

- Lớp (class) là sự trừu tượng hóa của đối tượng.
- Khác với kiểu dữ liệu thông thường, một lớp là một đơn vị (trừu tượng) bao gồm sự kết hợp giữa các phương thức và các thuộc tính.
- Hiểu nôm na hơn là các đối tượng có các đặc tính tương tự nhau được gom lại thành một lớp đối tượng.

Classes (3/3)

```
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;  
    }  
}
```

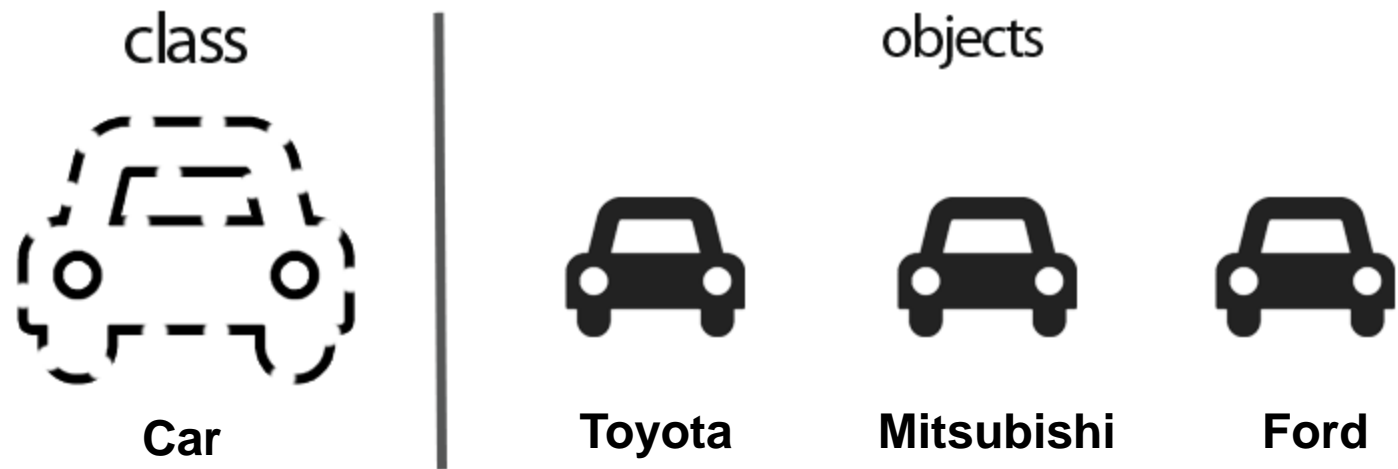
← Data field

← Constructors

← Method

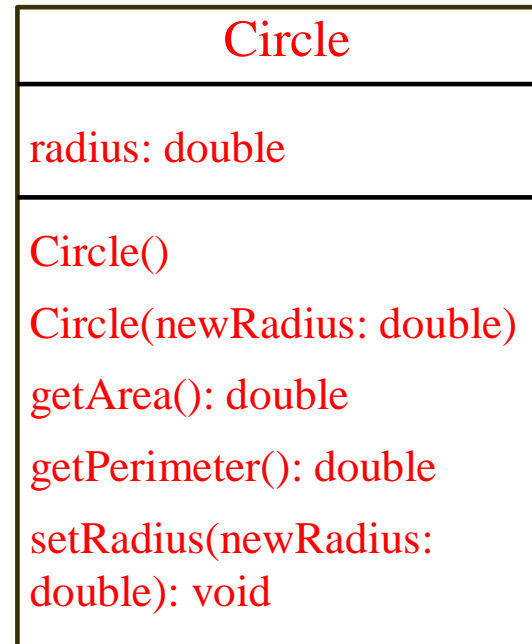
Objects vs Classes

- **Lớp** có thể hiểu như là *khuôn mẫu*, **đối tượng** là một *thực thể* thể hiện dựa trên khuôn mẫu đó.
- Ví dụ: Ta nói về xe ô tô con, ta có thể hiểu nó là class (lớp) xe ô tô có: Các đặc điểm (4 bánh...); Các hành động (Tiến, lùi, nháy đèn)
- Ví dụ: Đối tượng thì chính là xe Mitsubishi chúng ta chạy.



UML Class Diagram (1/5)

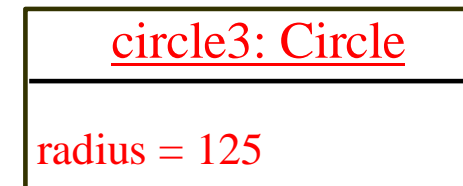
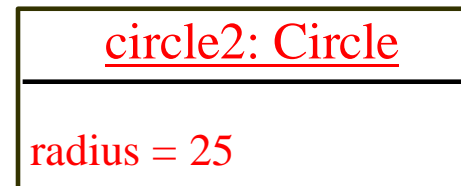
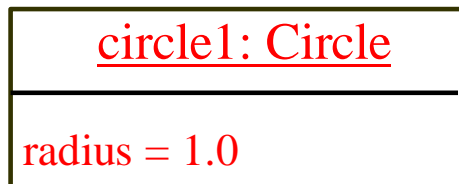
UML Class Diagram



← Class name

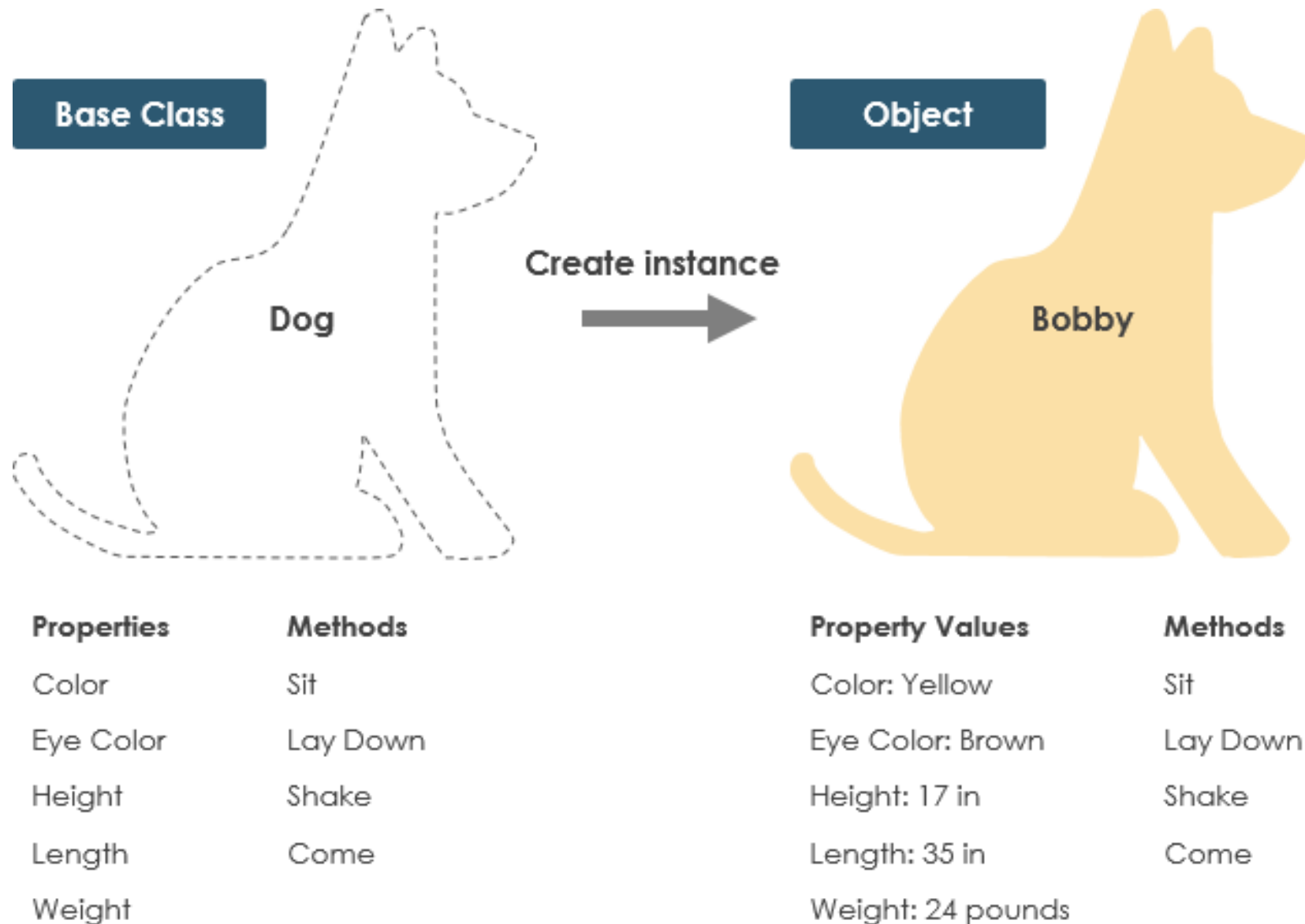
← Data fields

← Constructors and methods

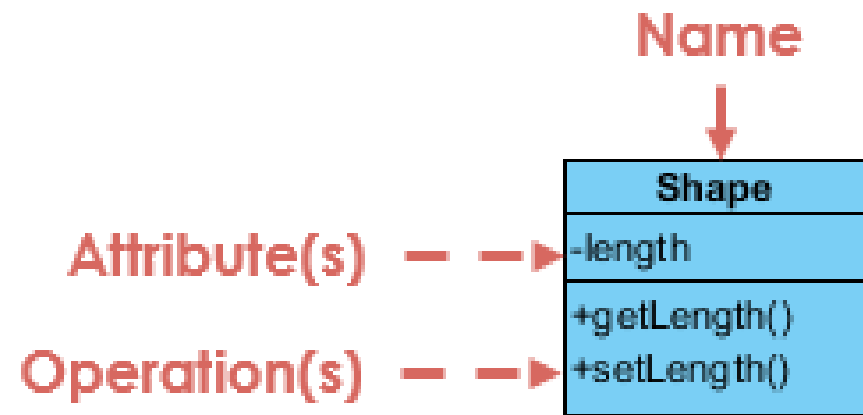


← UML notation for objects

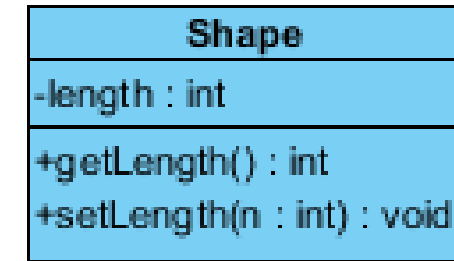
UML Class Diagram (2/5)



UML Class Diagram - Notation (3/5)

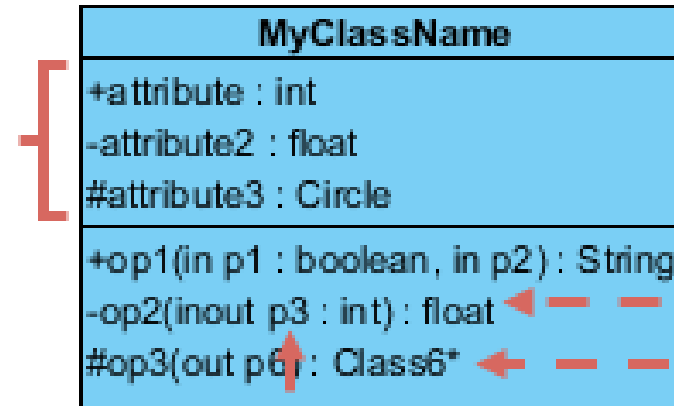


Class without signature



Class **with** signature

MyClassName has 3 attributes
and 3 operations



op2 returns a float

op3 returns a pointer
(denoted by a *) to Class6

Parameter p3 of op2 is of type int

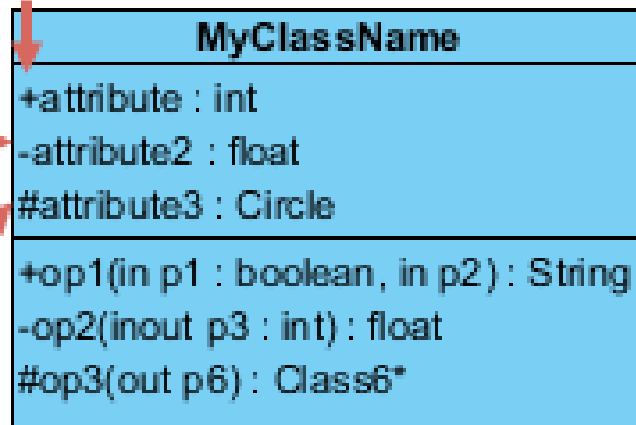
Các dấu (+) , (-) và (#) tương ứng với các thuộc tính (**public**), (**private**) và (**protected**).

UML Class Diagram - Notation (4/5)

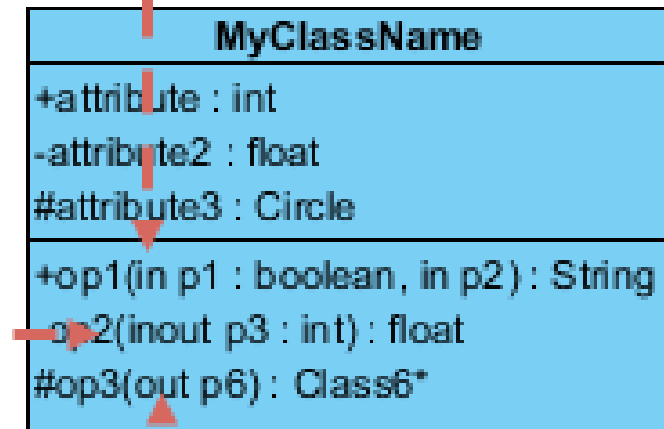
Public Attribute

Private Attribute

Protected Attributes



Passed to op1 by the caller

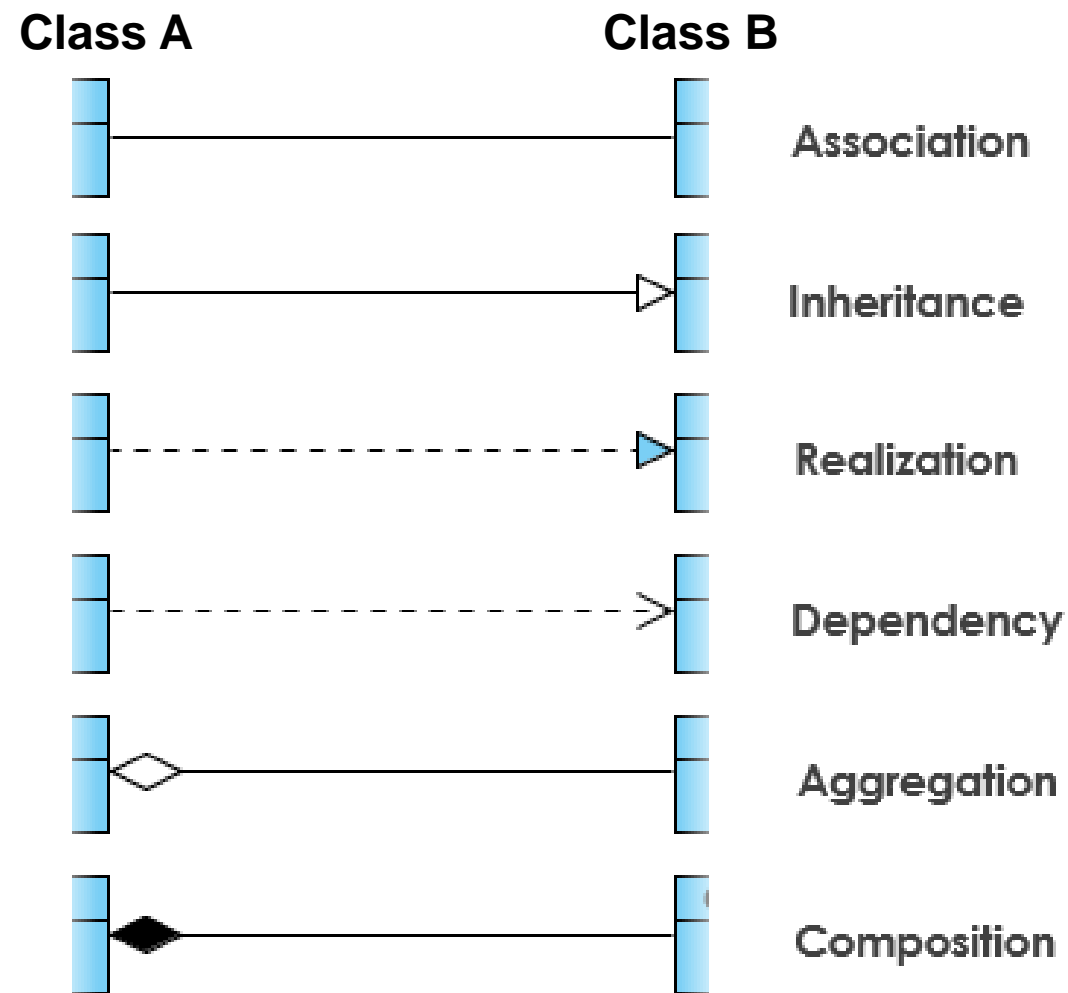


Passed to op2 by the caller, and possibly modified by op2 and is passed back

Not set by the caller but is modified by op3, and is passed back out

Mỗi tham số trong một hoạt động (**phương thức**) có thể được ký hiệu là vào, ra hoặc vào trong đó chỉ định hướng của nó đối với người gọi. Hướng này được hiển thị trước tên tham số.

UML Class Diagram – Relationship Types (5/5)



Aggregation cũng giống như Association, nhưng khác là Aggregation có mối quan hệ sở hữu (ownership) giữa các instance (Class A sở hữu Class B)

Association là sự liên kết giữa 2 class khi mà không cái nào sở hữu cái nào. Vòng đời các thể hiện của 2 class thì độc lập nhau và không có mối quan hệ sở hữu nào trong biểu diễn này.

Inheritance (or Generalization) Quan hệ tổng quát hóa. Hoặc quan hệ kế thừa Đối tượng cụ thể (concrete) sẽ kế thừa các thuộc tính và phương thức của đối tượng tổng quát (general). VD: B là cha của A, A là con của B.

Realization là một mối quan hệ giữa lớp kế hoạch chi tiết và đối tượng chứa các chi tiết mức độ thực hiện tương ứng của nó. Nói cách khác, ta có thể hiểu đây là mối quan hệ giữa *interface* và lớp *implementing*.

Dependency: Một đối tượng của một classA có thể sử dụng một đối tượng của một **class B** khác trong một phương thức của **class A**. Ta nói, **class A** phụ thuộc vào **class B**

Composition: Object của Class A có chứa object của Class B. Nếu A bị hủy thì B sẽ không tồn tại. ngược lại, B bị hủy thì sẽ không ảnh hưởng đến A.

Example: Defining Classes and Creating Objects

Dấu + biểu thị đây là một sửa đổi dạng công cộng (public modifier)



TV
channel: int volumeLevel: int on: boolean
+TV() +turnOn(): void +turnOff(): void +setChannel(newChannel: int): void +setVolume(newVolumeLevel: int): void +channelUp(): void +channelDown(): void +volumeUp(): void +volumeDown(): void

Kênh hiện tại (1 đến 120) của TV này.
Mức âm lượng hiện tại (1 đến 7) của TV.
Cho biết TV này đang bật / tắt.

Tạo đối tượng TV mặc định.

Bật TV này.

Tắt TV này.

Đặt kênh mới cho TV này.

Đặt mức âm lượng mới cho TV này.

Tăng số kênh lên 1.

Giảm số kênh đi 1.

Tăng mức âm lượng lên 1.

Giảm mức âm lượng đi 1.

Constructors

```
Circle() {  
}
```

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

Các *hàm tạo* (constructors) là một loại phương thức đặc biệt được gọi để *xây dựng* các đối tượng.

Constructors

- Một ***hàm tạo*** (constructor) không có tham số được gọi là một hàm tạo không đối số.
 - Các hàm tạo phải có ***cùng tên*** với chính lớp đó.
 - Các hàm tạo không có kiểu trả về - thậm chí không có cả từ khóa **void**.
 - Các hàm tạo được gọi bằng cách sử dụng toán tử **new** khi một đối tượng được tạo. Constructors đóng vai trò trong việc khởi tạo các đối tượng.

Constructors

- **Constructor** : Khởi tạo dữ liệu ban đầu cho các thuộc tính của đối tượng
- Constructor mặc định: khi không xây dựng constructor và các thuộc tính lấy giá trị mặc định
- Xây dựng constructor
 - Tên constructor trùng với tên của lớp
 - Tiền tố là public
 - Không có giá trị trả về
 - Có thể chồng constructor (có nhiều constructor)
- Sử dụng constructor : được gọi khi cấp phát 1 đối tượng cho lớp

Creating Objects Using Constructors

```
new ClassName();
```

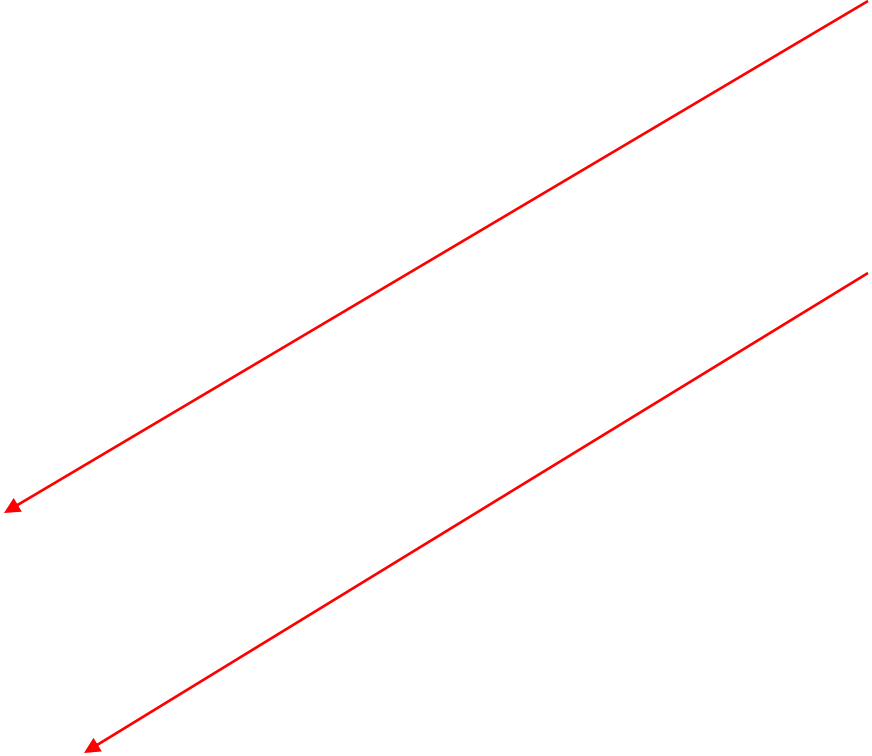
Ví dụ:

```
new Circle();
```

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

```
Circle() {  
}
```

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

Two red arrows originate from the yellow box containing the Circle class definition. One arrow points from the first constructor (Circle()) to the new Circle() statement. The other arrow points from the second constructor (Circle(double newRadius)) to the new Circle(5.0) statement.

Default Constructor

- Một lớp có thể được định nghĩa mà *không có hàm tạo*.
- Trong trường hợp này, một phương thức khởi tạo **no-arg** với phần thân trống được định nghĩa ngầm định trong lớp.
- Hàm tạo này, được gọi là *hàm tạo mặc định* (default constructor). Và nó chỉ được cung cấp tự động nếu *không có hàm tạo* nào được định nghĩa rõ ràng trong lớp.

Declaring Object Reference Variables

- Để tham chiếu một đối tượng, ta gán đối tượng cho một biến tham chiếu.
- Để khai báo một biến tham chiếu, hãy sử dụng cú pháp:

```
ClassName objectRefVar; //TênLớp biếnĐốiTượngThamChiếu
```

Ví dụ:

```
Circle myCircle;
```

Declaring/Creating Objects in a Single Step

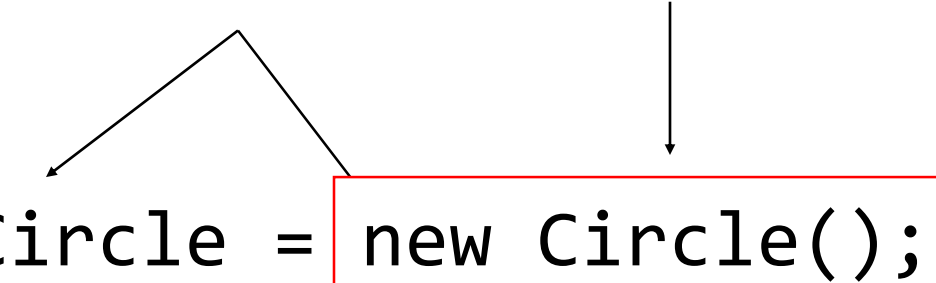
```
ClassName objectRefVar = new ClassName();
```

Assign object reference

Create an object

Ví dụ:

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



Accessing Object's Members

❑ Tham chiếu dữ liệu của đối tượng:

`objectRefVar.data`

Ví dụ: `myCircle.radius`

❑ Gọi phương thức của đối tượng:

`objectRefVar.methodName(arguments)`

Ví dụ: `myCircle.getArea()`

Trace Code

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

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

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

myCircle

No value

Khai báo *myCircle*

Trace Code

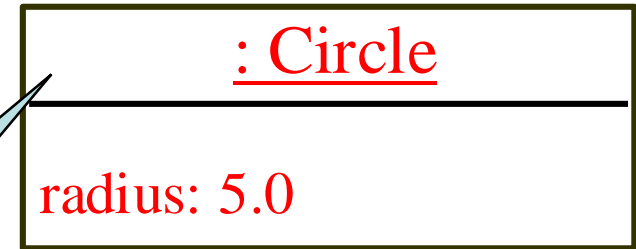
```
Circle myCircle = new Circle(5.0);
```

myCircle

No value

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

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



Tạo một circle
(vòng tròn)

Trace Code

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

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

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

myCircle

Reference value

: Circle

radius: 5.0

Gán đối tượng tham chiếu cho myCircle

Trace Code

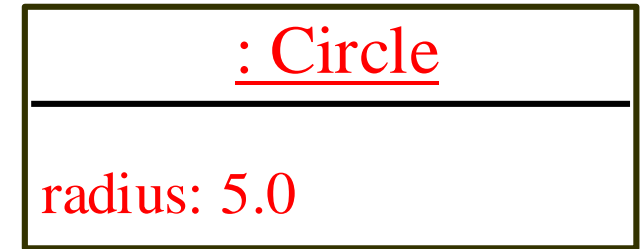
```
Circle myCircle = new Circle(5.0);
```

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

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

myCircle

Reference value



yourCircle

No value

Khởi tạo yourCircle

Trace Code

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

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

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

myCircle

Reference value



: Circle

radius: 5.0

yourCircle

No value

Tạo một đối tượng
Circle mới

: Circle

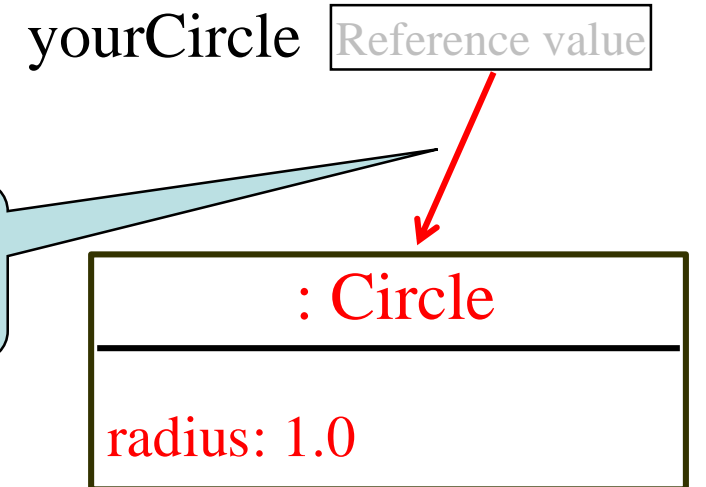
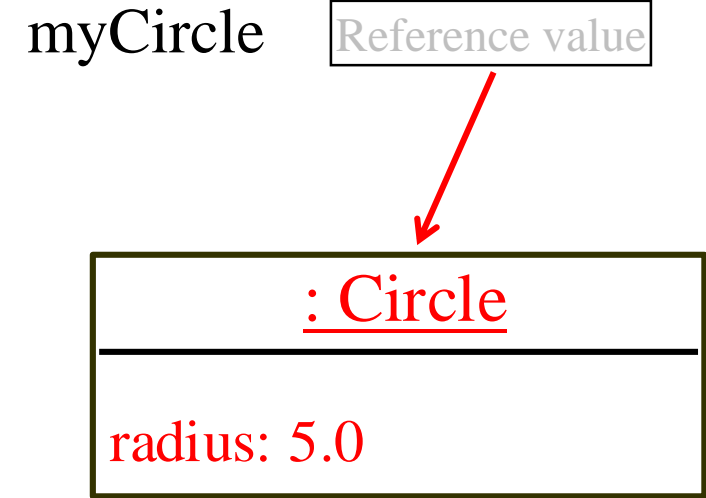
radius: 1.0

Trace Code

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

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

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



Gán đối tượng tham chiếu tới yourCircle

Trace Code

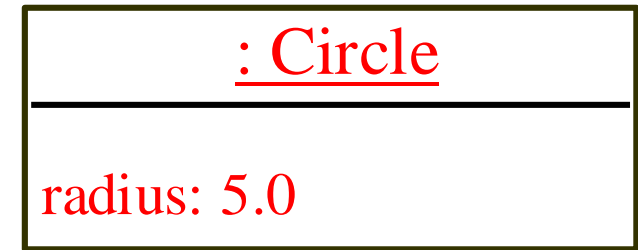
```
Circle myCircle = new Circle(5.0);
```

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

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

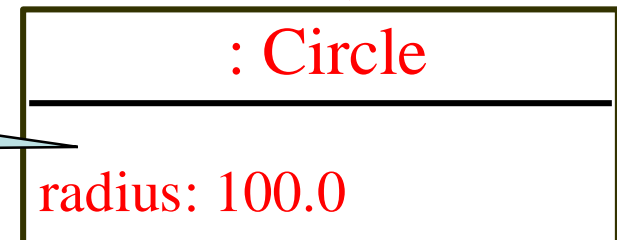
myCircle

Reference value



yourCircle

Reference value



Đổi giá trị của bán kính
(radius) trong yourCircle

Reference Data Fields

Các trường dữ liệu có thể là loại tham chiếu. Ví dụ, lớp Student sau đây chứa tên trường dữ liệu của kiểu Chuỗi.

```
public class Student {  
    String name; // name mặc định là null  
    int age; // age mặc định là 0  
    boolean isScienceMajor; // isScienceMajor mặc định là false  
    char gender; // c có giá trị mặc định là '\u0000'  
}
```

The null Value

- Nếu trường dữ liệu của kiểu tham chiếu không tham chiếu đến bất kỳ đối tượng nào, trường dữ liệu sẽ giữ một giá trị chữ đặc biệt, **null**.

Default Value for a Data Field

- Giá trị mặc định của trường dữ liệu là **null** đối với kiểu tham chiếu, **0** đối với kiểu số, **false** đối với kiểu boolean và **'\u0000'** đối với kiểu char. Tuy nhiên, Java không gán giá trị mặc định cho một biến cục bộ bên trong phương thức.

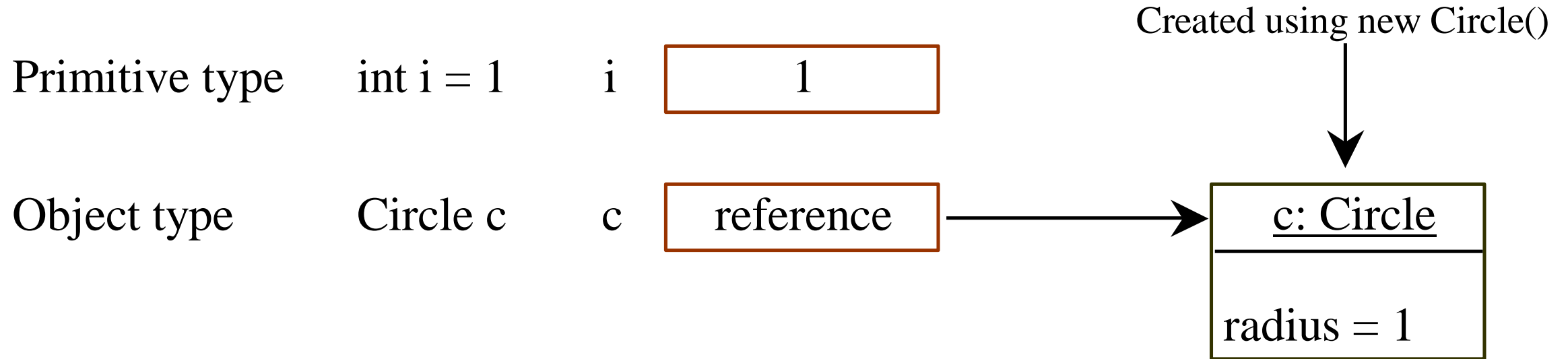
```
public class Test {  
    public static void main(String[] args) {  
        Student student = new Student();  
        System.out.println("name? " + student.name);  
        System.out.println("age? " + student.age);  
        System.out.println("isScienceMajor? " + student.isScienceMajor);  
 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


Example

- Java không gán giá trị mặc định nào cho một biến cục bộ bên trong một phương thức.

```
public class Test {  
    public static void main(String[] args) {  
        int x; // x không có giá trị mặc định  
        String y; // y không có giá trị mặc định  
        System.out.println("x is " + x);  
        System.out.println("y is " + y);  
    }  
}
```

Primitive Data Types vs Object Types



Copying Variables

Primitive type assignment $i = j$

Before:

i

1

j

2

After:

i

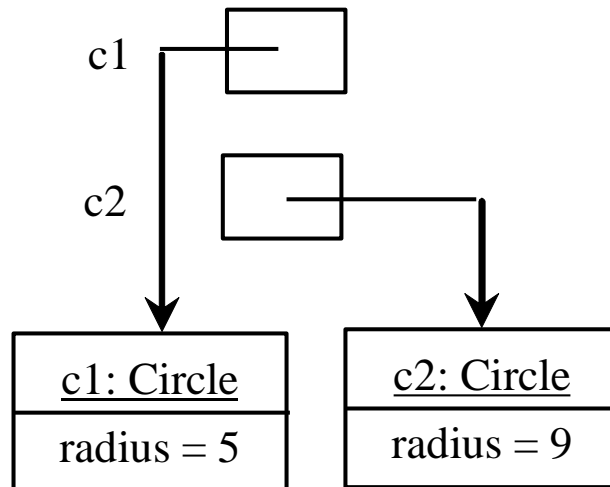
2

j

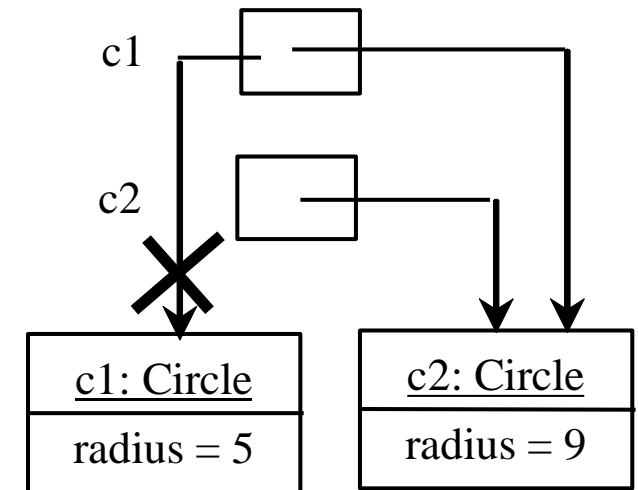
2

Object type assignment $c1 = c2$

Before:

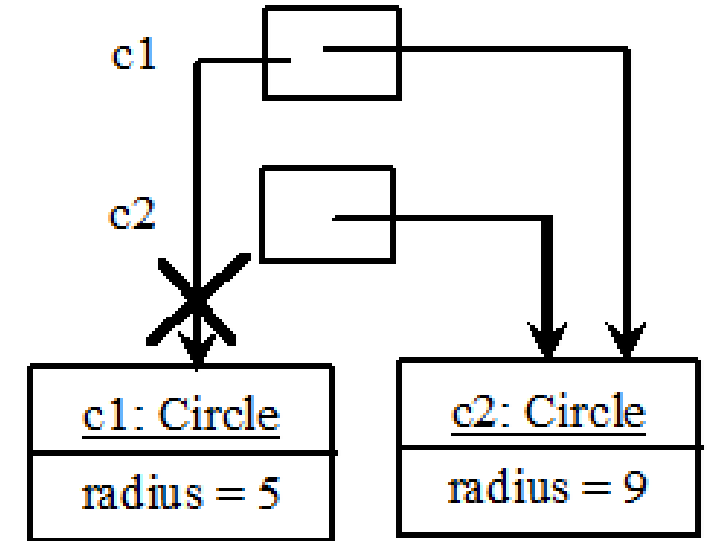


After:



Garbage Collection (1/2)

- Như trong hình trước, sau câu lệnh gán $c1 = c2$, $c1$ trỏ đến cùng một đối tượng được tham chiếu bởi $c2$.
- Đối tượng được tham chiếu trước đó bởi $c1$ không còn được tham chiếu nữa. Đối tượng này được gọi là rác.
- Rác được JVM thu gom tự động.



Garbage Collection (2/2)

- ❑ **TIP:** Nếu ta biết rằng một đối tượng không còn cần thiết nữa, ta có thể gán rõ ràng **null** cho một biến tham chiếu cho đối tượng.
 - JVM sẽ tự động thu thập không gian lưu trữ nếu đối tượng không được tham chiếu bởi bất kỳ biến nào.

The Date Class

- Java cung cấp một gói ngày và giờ độc lập với hệ thống trong lớp **java.util.Date**. Bạn có thể sử dụng lớp **Date** để tạo một instance cho **date** và **time** hiện tại và sử dụng phương thức **toString** của nó để trả về **date** và **time** dưới dạng một chuỗi (**String**).

The + sign indicates
public modifier



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

Xây dựng một đối tượng Date cho thời gian hiện tại.

Tạo đối tượng Date trong một thời gian nhất định tính bằng mili giây đã trôi qua kể từ ngày 1 tháng 1 năm 1970, giờ GMT.

Trả về một chuỗi đại diện cho ngày và giờ.

Trả về số mili giây kể từ ngày 1 tháng 1 năm 1970, giờ GMT.

Đặt thời gian trôi qua mới trong đối tượng.

The Date Class Example

Ví dụ, đoạn mã sau:

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

hiển thị một chuỗi như Sun Mar 09 13:50:19 EST 2003.

The Random Class

- Ta đã sử dụng `Math.random()` để nhận giá trị kép ngẫu nhiên giữa 0.0 và 1.0 (không bao gồm 1.0). Một trình tạo số ngẫu nhiên hữu ích hơn được cung cấp trong lớp `java.util.Random`.

java.util.Random	
+Random()	Xây dựng một đối tượng Random với thời gian hiện tại làm hạt giống của nó.
+Random(seed: long)	Xây dựng một đối tượng Random với một hạt giống xác định.
+nextInt(): int	Trả về một giá trị int ngẫu nhiên.
+nextInt(n: int): int	Trả về một giá trị int ngẫu nhiên giữa 0 và n (không bao gồm).
+nextLong(): long	Trả về một giá trị dài ngẫu nhiên.
+nextDouble(): double	Trả về một giá trị kép ngẫu nhiên giữa 0,0 và 1,0 (không bao gồm).
+nextFloat(): float	Trả về giá trị thực ngẫu nhiên giữa 0,0F và 1,0F (không bao gồm).
+nextBoolean(): boolean	Trả về một giá trị boolean ngẫu nhiên.

The Random Class Example

- Nếu hai đối tượng **Random** có cùng hạt giống, chúng sẽ tạo ra các dãy số giống hệt nhau. Ví dụ, đoạn mã sau đây tạo ra hai đối tượng **Random** có cùng hạt giống 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
```

The Point2D Class

- Java API có một lớp **Point2D** tiện lợi trong gói **javafx.geometry** để biểu diễn một điểm trong mặt phẳng hai chiều.

javafx.geometry.Point2D

```
+Point2D(x: double, y: double)
+distance(x: double, y: double): double
+distance(p: Point2D): double
+getX(): double
+getY(): double
+toString(): String
```

Constructs a `Point2D` object with the specified *x*- and *y*-coordinates.
Returns the distance between this point and the specified point (*x*, *y*).
Returns the distance between this point and the specified point *p*.
Returns the *x*-coordinate from this point.
Returns the *y*-coordinate from this point.
Returns a string representation for the point.

Instance Variables, and Methods

- Các biến cá thể (**instance** variable) thuộc về một cá thể (**instance**) cụ thể.
- Các *phương thức instance* được gọi bởi một **instance** của lớp.

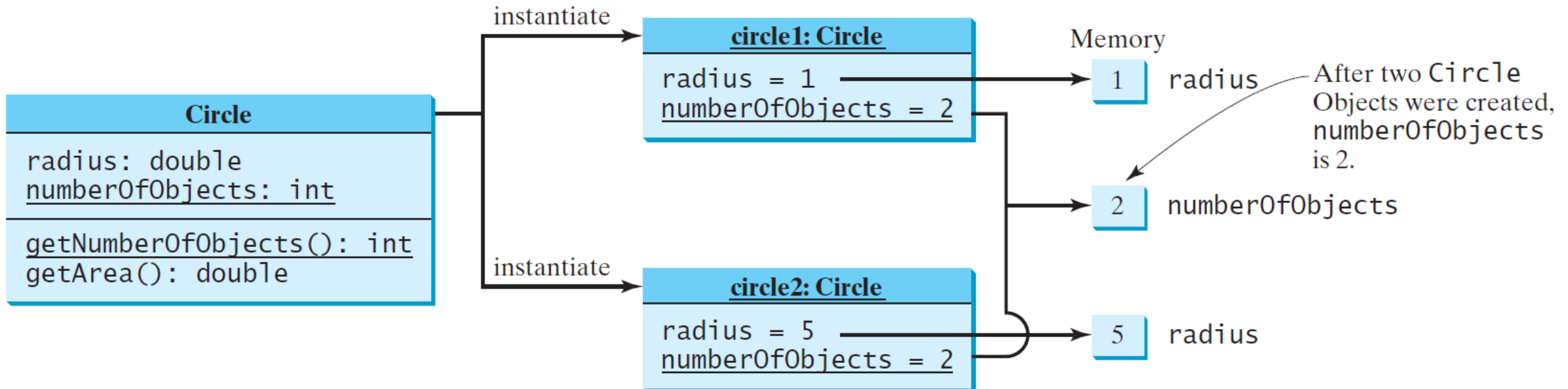
Static Variables, Constants, and Methods

- Các **biến tĩnh** (static variable) được dung chung bởi tất cả các **instances** của lớp.
- Các **phương thức tĩnh** (static method) không bị ràng buộc với một đối tượng cụ thể.
- **Hằng số tĩnh** (static constant) là các biến **final** được dung chung bởi tất cả các **instances** của lớp.

Static Variables, Constants, and Methods

- Để khai báo các biến, hằng và phương thức tĩnh, ta sử dụng công cụ sửa đổi tĩnh (**static modifier**).

UML Notation:
underline: static variables or methods



Visibility Modifiers and Accessor/Mutator Methods

Mặc định, lớp, biến hoặc phương thức có thể được truy cập bởi bất kỳ lớp nào trong cùng một gói (package).

❑ `public`

- Lớp, dữ liệu hoặc phương thức được dùng cho bất kỳ lớp nào trong bất kỳ đâu trong chương trình (hoặc package).

❑ `private`

- Dữ liệu hoặc phương thức chỉ có thể được truy cập bởi lớp khai báo.
- Phương thức **get** và **set** được sử dụng để đọc và sửa đổi các thuộc tính private.

Visibility Modifiers and Accessor/Mutator Methods

Private modifier hạn chế quyền truy cập trong một lớp, **modifier** mặc định hạn chế quyền truy cập trong một gói và **modifier** công khai cho phép truy cập không hạn chế.

```
package p1;

public class C1 {
    public int x;
    int y;
    private int z;

    public void m1() {
    }
    void m2() {
    }
    private void m3() {
    }
}
```

```
package p1;

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();
    }
}
```

Visibility Modifiers and Accessor/Mutator Methods

```
package p1;
```

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

```
package p1;
```

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

```
package p2;
```

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

Modifier mặc định trong một lớp hạn chế quyền truy cập trong một gói và **modifier** công khai cho phép truy cập không hạn chế.

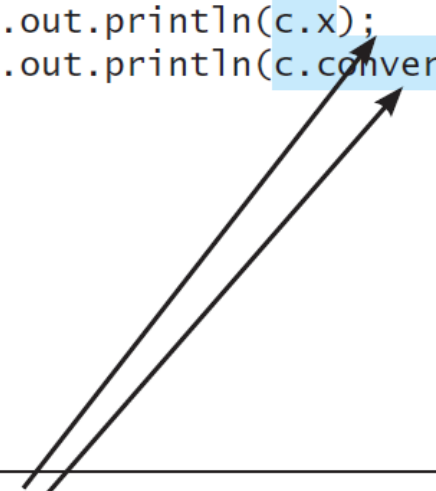
NOTE

- Một đối tượng không thể truy cập các thành viên riêng tư (private) của nó, xem hình (b). Tuy nhiên, sẽ là OK nếu đối tượng được khai báo trong lớp riêng của nó, xem hình (a).

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

(a) This is okay because object `c` is used inside the class `C`.

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



(b) This is wrong because `x` and `convert` are private in class `C`.

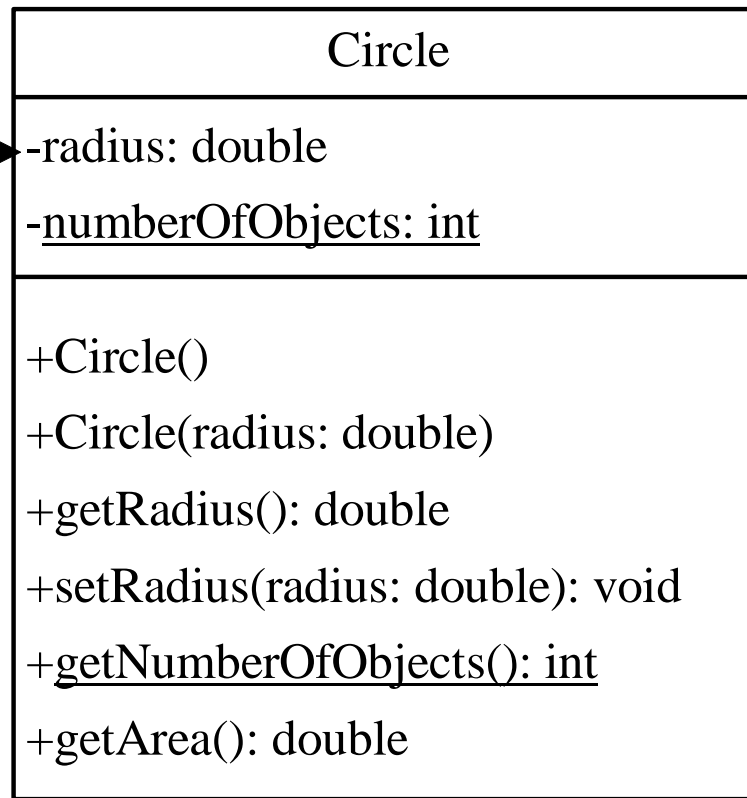
Why Data Fields Should Be private?

- ❑ Để bảo vệ dữ liệu.
- ❑ Để làm cho mã nguồn dễ bảo trì.

Example of Data Field Encapsulation

- Ví dụ về đóng gói trường dữ liệu

The - sign indicates
private modifier



Bán kính của hình tròn (mặc định: 1,0).

Số lượng đối tượng vòng tròn được tạo.

Xây dựng một đối tượng vòng tròn mặc định.

Xây dựng một đối tượng hình tròn với bán kính xác định.

Trả về bán kính của hình tròn.

Đặt bán kính mới cho vòng tròn.

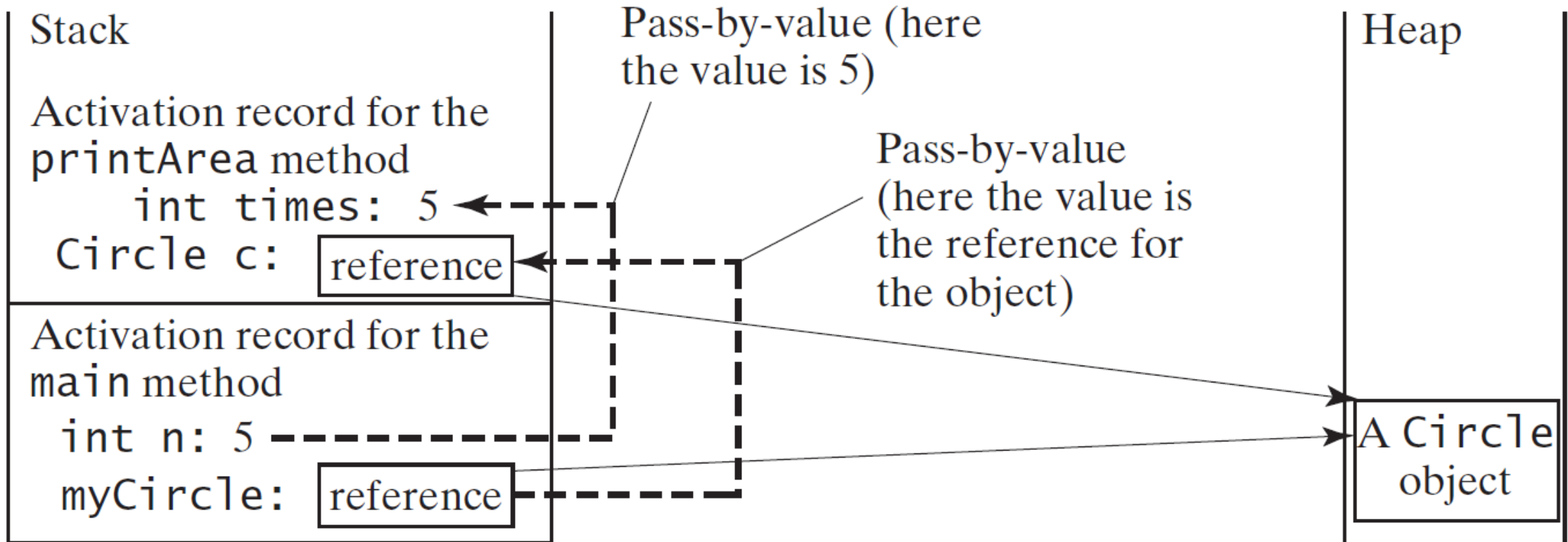
Trả về số đối tượng vòng tròn được tạo.

Trả về diện tích của hình tròn.

Passing Objects to Methods (1/2)

- ❑ Truyền theo giá trị cho giá trị kiểu nguyên thủy (giá trị được truyền cho tham số)
- ❑ Truyền theo giá trị cho giá trị kiểu tham chiếu (giá trị là tham chiếu đến đối tượng)

Passing Objects to Methods (2/2)



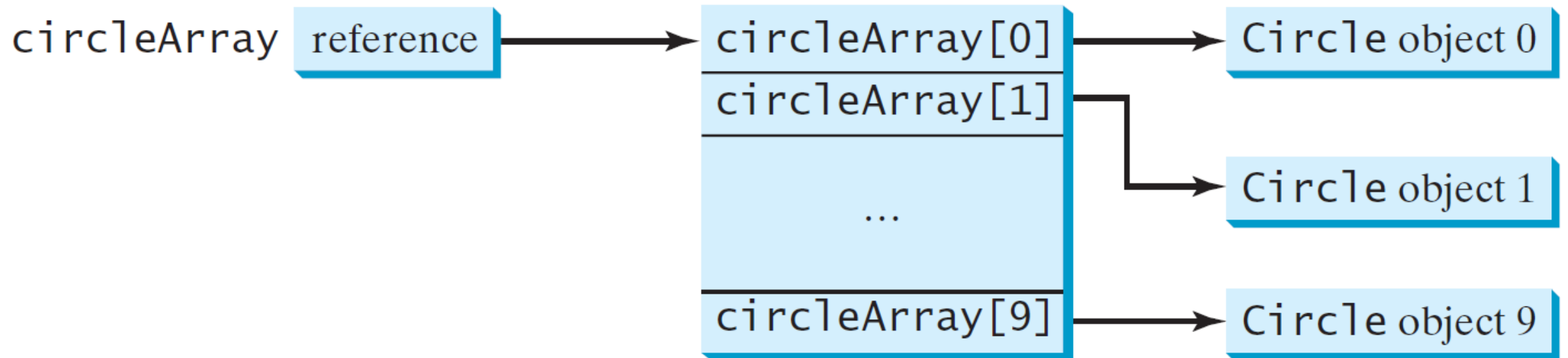
Array of Objects

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

- Một mảng các đối tượng thực ra là một mảng các **biến tham chiếu**.
- Vì vậy, việc gọi **circleArray [1].getArea()** liên quan đến hai cấp độ tham chiếu như thể hiện trong hình ở slide sau.
 - **circleArray** tham chiếu đến toàn bộ mảng.
 - **circleArray [1]** tham chiếu đến một đối tượng **Circle**.

Array of Objects

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



Immutable Objects and Classes

- Nếu nội dung của một đối tượng không thể thay đổi sau khi đối tượng được tạo, đối tượng đó được gọi là đối tượng bất biến (**immutable object**) và lớp của nó được gọi là lớp bất biến (**immutable class**).
- Nếu ta xóa phương thức set trong lớp Circle lớp này sẽ là bất biến vì **radius** là private và không thể thay đổi nếu không có phương thức set.

Circle
-radius: double - <u>numberOfObjects</u> : int
+Circle() +Circle(radius: double) +getRadius(): double +setRadius(radius: double): void + <u>getNumberOfObjects</u> (): int +getArea(): double

Example

```
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;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Student student = new Student(111223333, 1970, 5, 3);
        BirthDate date = student.getBirthDate();
        date.setYear(2010); // Bây giờ năm sinh của học sinh được thay đổi!
    }
}
```

What Class is Immutable?

- Để một lớp là *Immutable*, nó phải đánh dấu tất cả các trường dữ liệu là riêng tư (private) và:
 - Không cung cấp phương thức trình đột biến;
 - Không có phương thức trình truy cập nào sẽ trả về tham chiếu đến đối tượng trường dữ liệu có thể thay đổi (mutable).

Scope of Variables

- ❑ Phạm vi của biến instance và biến tstatic là toàn bộ lớp. Chúng có thể được khai báo ở bất kỳ đâu bên trong một lớp.
- ❑ Phạm vi của một biến cục bộ bắt đầu từ khai báo của nó và tiếp tục đến cuối khối chứa biến. Biến cục bộ phải được khởi tạo một cách rõ ràng trước khi nó có thể được sử dụng.

The this Keyword

- ❑ Từ khóa **this** là tên của một tham chiếu đề cập đến một đối tượng chính nó (itself). Cách sử dụng phổ biến của từ khóa này là tham chiếu đến các **trường dữ liệu ẩn** (*hidden data field*) của một lớp.
- ❑ Một cách sử dụng phổ biến khác của từ khóa **this** là cho phép một phương thức khởi tạo (constructor) gọi một phương thức khởi tạo khác trong cùng một lớp.

Reference the Hidden Data Fields

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

Giả sử f1 và f2 là hai đối tượng của F.

```
f1 = new F(); F f2 = new F();
```

Gọi **f1.setI(10)** là để thực thi

```
this.i = 10, where this refers f1
```

Gọi **f2.setI(45)** là để thực thi

```
this.i = 45, where this refers f2
```

Calling Overloaded Constructor

Gọi một nạp
chồng của hàm
khởi tạo

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

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

Đây phải được sử dụng một cách rõ ràng để tham chiếu đến bán kính trường dữ liệu của đối tượng đang được xây dựng

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

Đây được sử dụng để gọi một phương thức khởi tạo khác

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

Mọi biến instance thuộc về một instance được đại diện bởi điều này, tuy nhiên ta thường được bỏ qua

Inner Class

- Inner Class (Lớp nội) là lớp được khai báo bên trong 1 lớp khác.
- Lớp nội có thể truy xuất trực tiếp biến của lớp cha.
- Trong ví dụ trên thì chương trình dịch sẽ tạo ra hai lớp với hai files khác nhau: **A.class** và **B.class**

```
public class A {  
    // ...  
    int <field_1>  
    class B {  
        // ...  
        int <field_2>  
        public B(int par_1) {  
            field_2 = par_1 + field_1;  
        }  
    }  
}
```

Quizz!

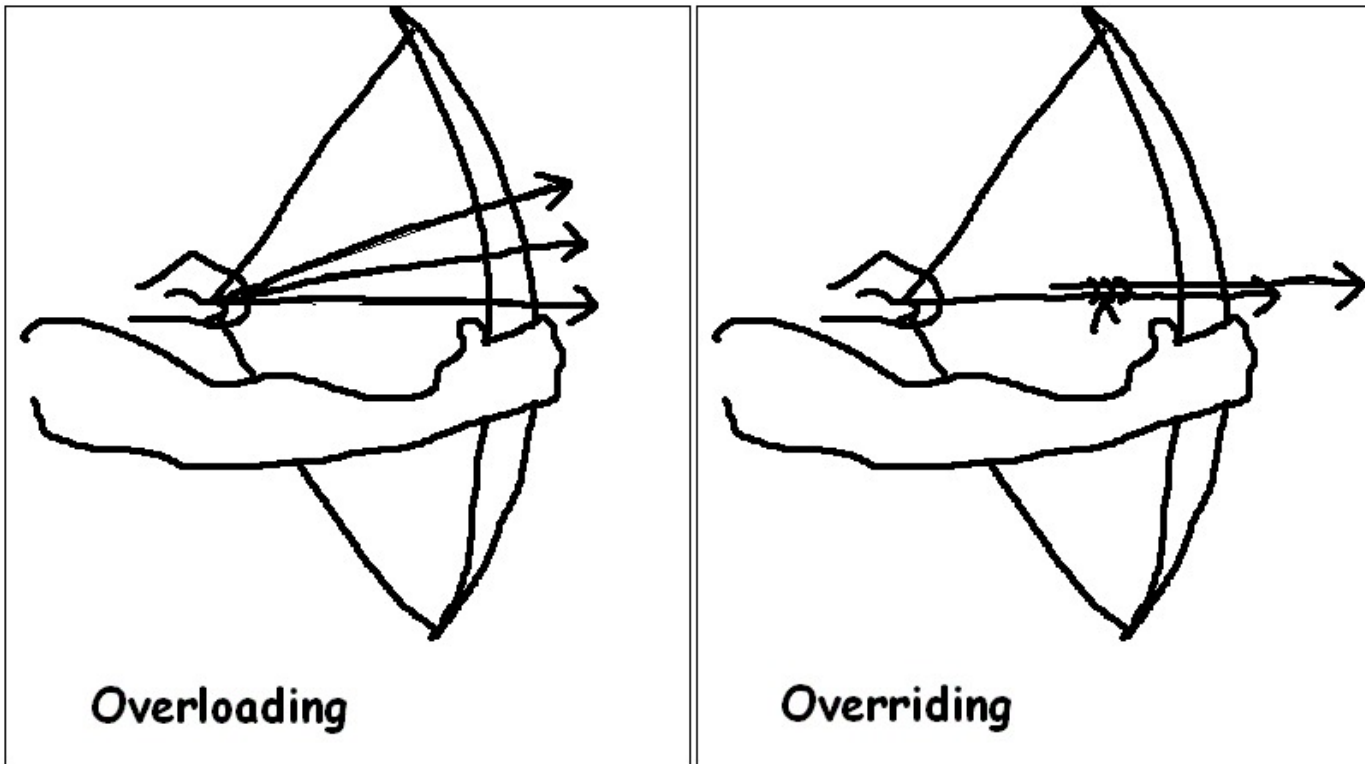
- 1) Bạn có thể nói gì về Constructor?
- 2) Liệt kê ba bước để tạo một đối tượng cho một lớp?
- 3) Phương thức finalize() làm gì?
- 4) Bạn hiểu gì về Đối tượng?

Exercise

- 1) Xây dựng lớp PhanSo với 2 thuộc tính tử, mẫu và đầy đủ các phép toán trên phân số
- 2) Xây dựng lớp SoPhuc với 2 thuộc tính thực, ảo, và đầy đủ các phép toán trên phân số
- 3) Xây dựng lớp Bac1 gồm 2 thuộc tính hệ số 1, hệ số 0 và phương thức giải phương trình bậc 1
- 4) Xây dựng lớp Bac2 gồm 3 thuộc tính hệ số 2, hệ số 1, hệ số 0 và phương thức giải phương trình bậc 2
- 5) Xây dựng lớp mảng số nguyên gồm thuộc tính là mảng số nguyên và số phần tử trong mảng cùng một số thao tác trên mảng sau: thêm 1 giá trị vào chỉ số i, thêm 1 giá trị vào cuối mảng, xóa, sắp xếp tăng, tìm kiếm 1 giá trị x, lấy giá trị tại chỉ số i, xác định số phần tử của mảng & ghi đè giá trị x vào chỉ số i

Youtube

- <https://www.youtube.com/watch?v=8yjkWGRlUmY>



Q&A

