



Northeastern  
University

# Lecture 6: Object Oriented Programming - 1

Prof. Chen-Hsiang (Jones) Yu, Ph.D.  
College of Engineering

Materials are edited by Prof. Jones Yu from

Liang, Y. Daniel. Introduction to Java Programming and Data Structures, Comprehensive  
Version, 12th edition, Pearson, 2019.

# Outline

---

- Objects and Classes
- Thinking in Objects

# Outline

---

- Objects and Classes
- Thinking in Objects

# Objects and Classes

Objects

# 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, such as a desk, a button, a car, etc.
- An object has **a unique identity**, **state**, and **behaviors**.

# OO Programming Concepts

---

- 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*.

# Objects

---

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



# Classes

# Classes

---

- *Classes* are constructs (structures, blueprints) that define objects of the same type.
- A Java class uses variables to define states (data fields) and methods to define behaviors.
- Additionally, a class provides a special type of methods, known as constructors, which are called to construct (initialize) objects from the class.

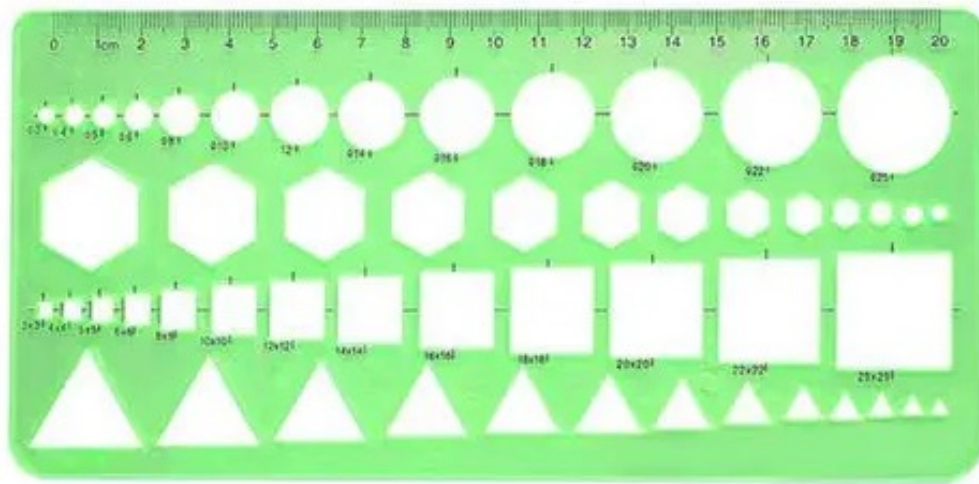


image source: <https://tinyurl.com/37h6h43w>, <https://tinyurl.com/2cxc3dc8>

# 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;  
    }  
}
```

← Data field

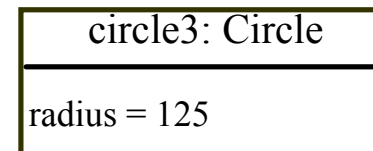
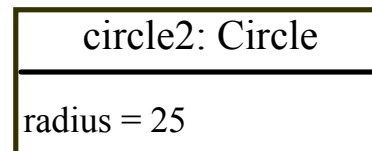
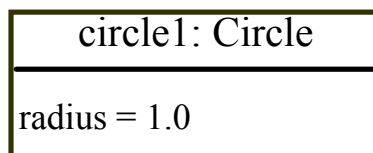
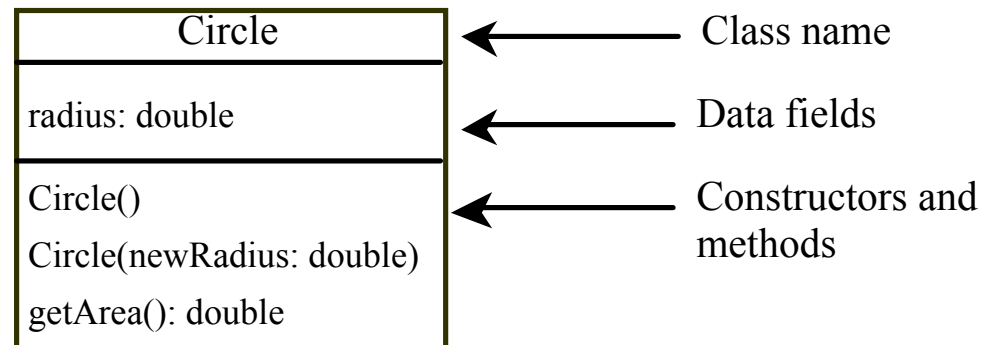
← Constructors

← Method

# UML (Unified Modeling Language) Class Diagram

---

UML Class Diagram



← UML notation  
for objects

# Exercise

---

- Create a new Java project. (“L6\_E1”)
- Create and define a TV class (“TV.java”) to have following data fields and methods:
  - » data fields:
    - channel (int)
    - volumeLevel (int)
    - on (boolean)
  - » methods:
    - turnOn(): void
    - turnOff(): void

# Answer

---

```
public class TV {  
  
    int channel;  
    int volumeLevel;  
    boolean on;  
  
    TV() {  
        channel = 1;  
        volumeLevel = 1;  
        on = false;  
    }  
  
    public void turnOn() {  
        on = true;  
    }  
  
    public void turnOff() {  
        on = false;  
    }  
  
}
```

```
public class TV {  
  
    int channel = 1;  
    int volumeLevel = 1;  
    boolean on = false;  
  
    public void turnOn() {  
        on = true;  
    }  
  
    public void turnOff() {  
        on = false;  
    }  
  
}
```

TV.java (two possible solutions)

# Constructing Objects Using Constructors



# Constructors

---

- **Constructors** are a special kind of methods that are called to construct (initialize) objects.

```
Circle() {  
}
```

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

# Constructors

---

- A constructor with no parameters is referred to as a *no-arg constructor* (or *default constructor*)
  - » Constructors must have the same name as the class itself.
  - » Constructors do not have a return type - not even void.
  - » Constructors are called 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);
```

```
class Circle {
```

```
...
```

```
Circle() {  
}
```

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

```
...
```

```
}
```

# Default Constructors

---

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

## Accessing Objects via Reference Variables

# Declaring object Reference Variables

---

- To reference an object, assign **the object** to a **reference variable**.
- 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:

Assign object reference      Create an object

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

# Accessing Object's Members

---

- 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;

myCircle

no value

Declare myCircle

# Trace Code

---

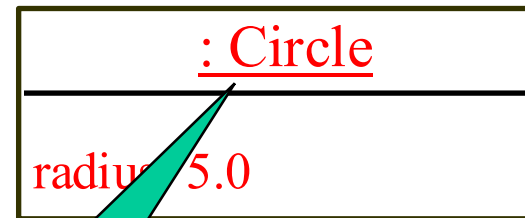
Circle myCircle = **new Circle(5.0);**

myCircle

no value

Circle yourCircle = new Circle();

yourCircle.radius = 100;



Create a circle

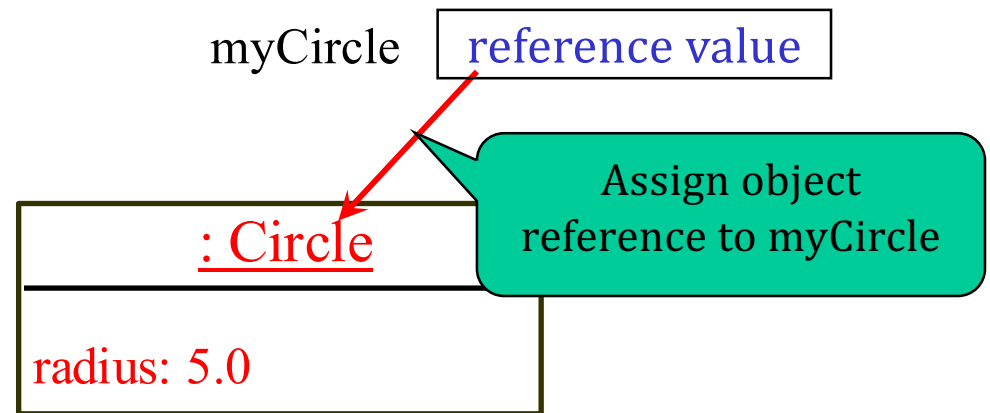
# Trace Code

---

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

**Circle yourCircle = new Circle();**

**yourCircle.radius = 100;**



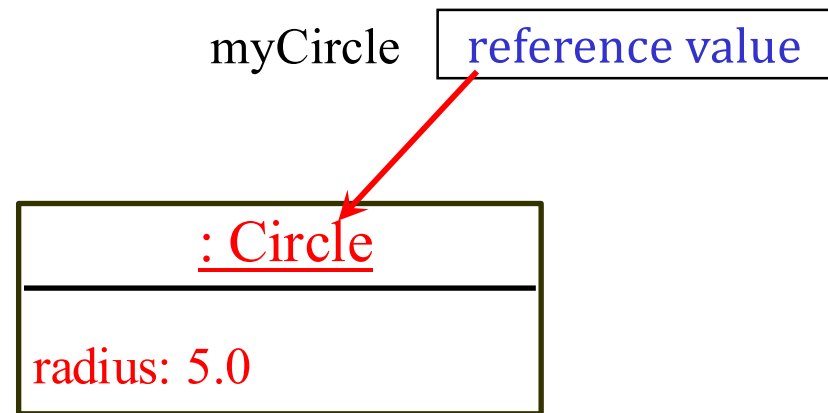
# Trace Code

---

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

**Circle yourCircle = new Circle();**

**yourCircle.radius = 100;**



yourCircle

no value

Declare 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

: Circle

radius: 1.0

Create a new  
Circle object

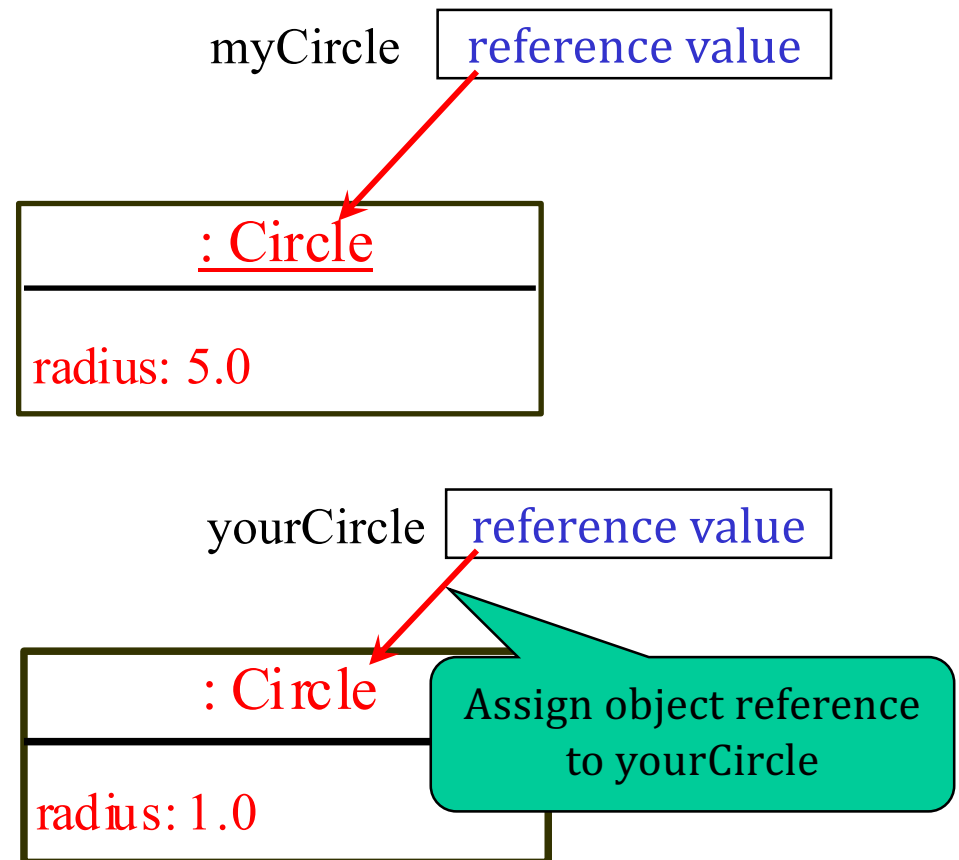
# Trace Code

---

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

**Circle yourCircle = new Circle();**

**yourCircle.radius = 100;**



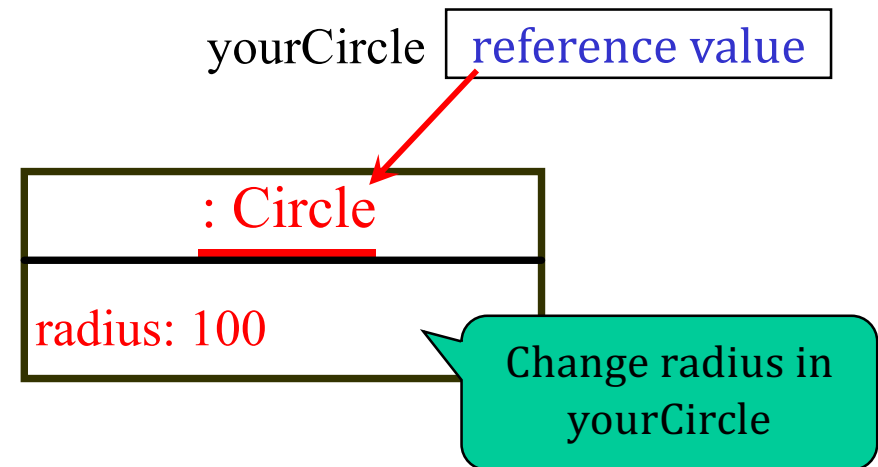
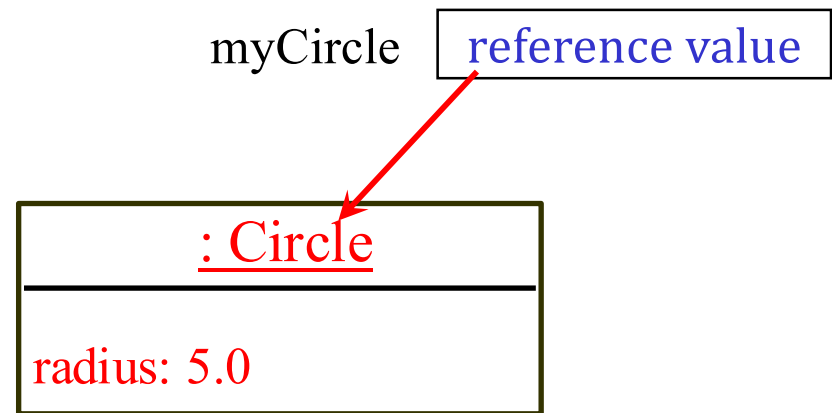
# Trace Code

---

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

**Circle yourCircle = new Circle();**

**yourCircle.radius = 100;**





# Default Value for a Data Field

---


- The default value of a data field:
  - » null for a reference type
  - » 0 for a numeric type
  - » false for a boolean type
  - » '\u0000' (null character) for a char type
- However, Java assigns no default value to a local variable inside a method.

# 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);  
    }  
}
```



Compile error: variable  
not initialized

# Exercise

---

- Define a `Student` class to have following data fields:  
`name` (String), `age` (int), `isScienceMajor` (boolean) and `gender` (char)
- Write a program (a.k.a. a driver program) to create an object of `Student` class type and `print out the default values of its data fields`.

# Answer

---

```
public class Student {  
    String name;  
    int age;  
    boolean isScienceMajor;  
    char gender;  
}
```

Student.java

```
public class ClassExamples {  
    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);  
    }  
}
```

ClassExamples.java

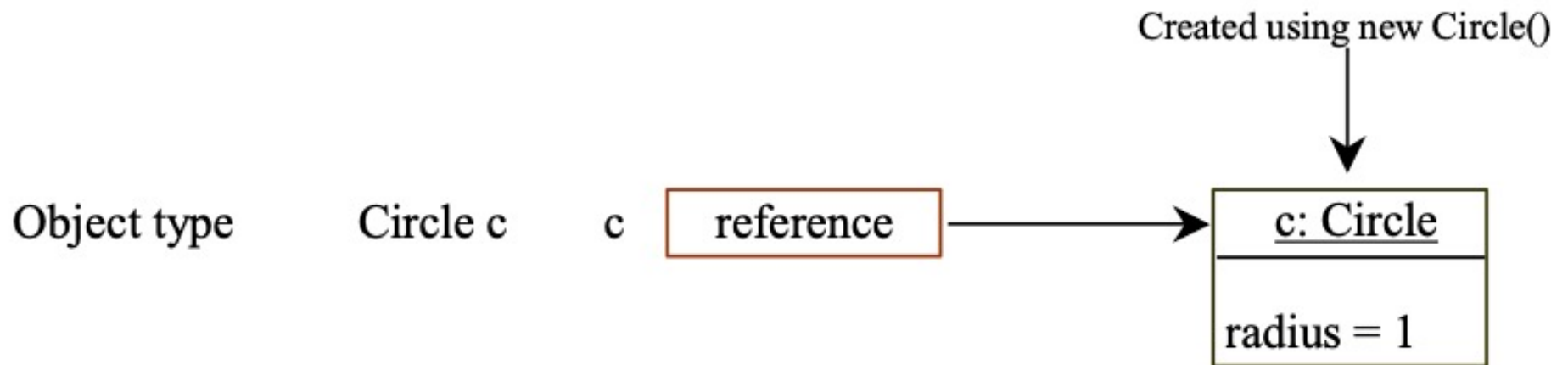
```
name? null  
age? 0  
isScienceMajor? false  
gender?
```

Output

# Primitive Data Type vs. Object Types

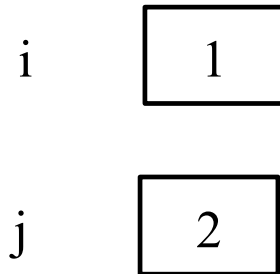
---

Primitive type    `int i = 1`    `i`    1

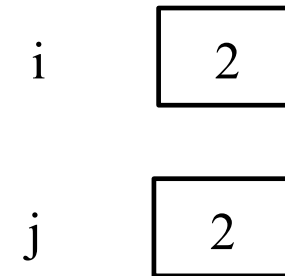


## Primitive type assignment $i = j$

Before:

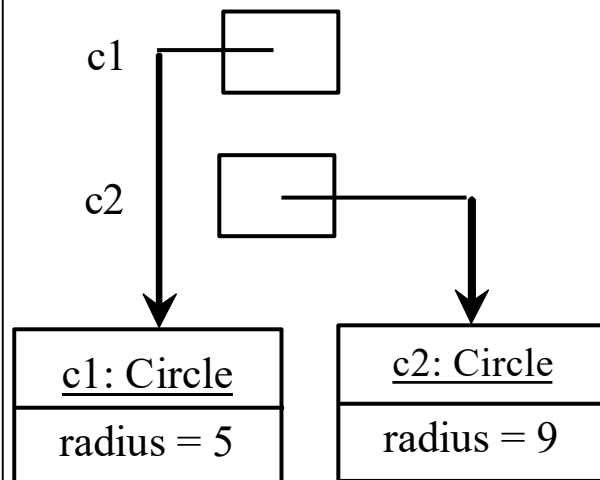


After:

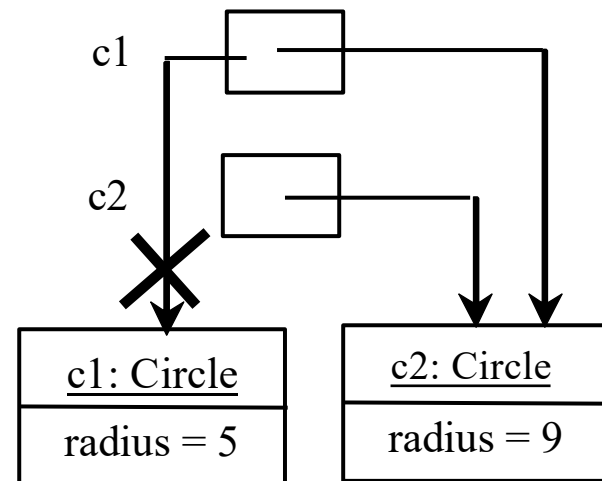


## Object type assignment $c1 = c2$

Before:



After:



# Garbage Collections

---

- 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.

# Garbage Collections

---

- 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.