Objects and Classes

CPT204 Advanced Object-Oriented Programming

Lecture 1.2 OOP Review 1

Contents

- Motivating Problems: Complex objects and GUIs
- Classes, objects, object state and behavior
- Object-oriented Design
- Constructors
- Accessing fields and methods
- Static vs. Non-static
- Static Variables and Methods
- Default values for Class Fields
- Primitive Data Types and Object Types, Copying
- Garbage Collection
- Example classes in Java API: the Date class
- The Random class
- Visibility Modifiers and Accessor/Mutator Methods

Object-Oriented Programming Concepts

- An object represents an entity in the real world that can be distinctly identified from a class/templates of objects with common properties.
- An object has a unique state and behavior:
  - the state of an object consists of a set of data fields (properties) with their current values
  - the behavior of an object is defined by a set of instance methods

Classes

- In Java classes are templates that define objects of the same type
  - A Java class uses:
    - non-static/instance variables to define data fields
    - non-static/instance methods to define behaviors
  - A class provides a special type of methods called ***constructors*** which are invoked to construct objects from the class

Classes

```
class Circle {
  /** The radius of this circle */
  private double radius = 1.0;

  /** Construct a circle object */
  public Circle() {
  }

  /** Construct a circle object */
  public Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  public double getArea() {
    return radius * radius * 3.14159;
  }
}
```

Data field

Constructors

Method

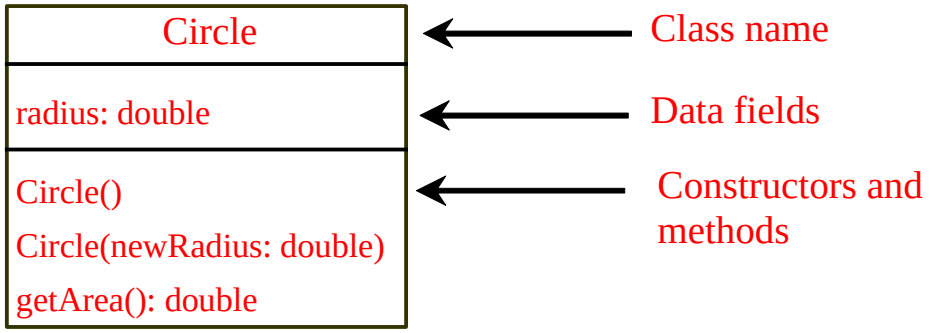
Classes

```
public class TestCircle {

  public static void main(String[] args) {

    Circle c1 = new Circle();
    Circle c2 = new Circle(5.0);

    System.out.println( c1.getArea() );
    System.out.println( c2.getArea() );

  }

}
```

Object-oriented Design

- The ***Unified Modeling Language*** (***UML***) is a general-purpose modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a object-oriented system.

UML Class Diagram



Constructors

- 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 – they initialize objects to **reference variables**:

```
ClassName o = new ClassName();
```

  - Example:

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

- A class may be declared without constructors: a no-arg **default constructor** with an empty body is **implicitly** declared in the class

Accessing Objects

- Referencing the object's data:

  `objectRefVar.data`

  - Example: **`myCircle.radius`**

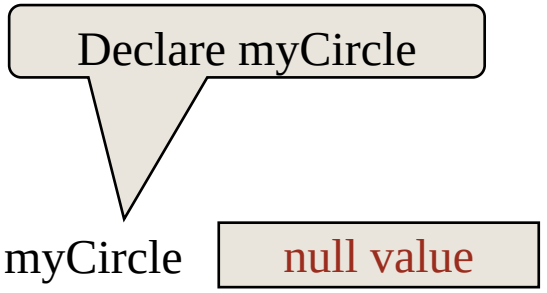
- Invoking the object's method:

`objectRefVar.methodName(arguments)`

  - Example: **`myCircle.getArea()`**

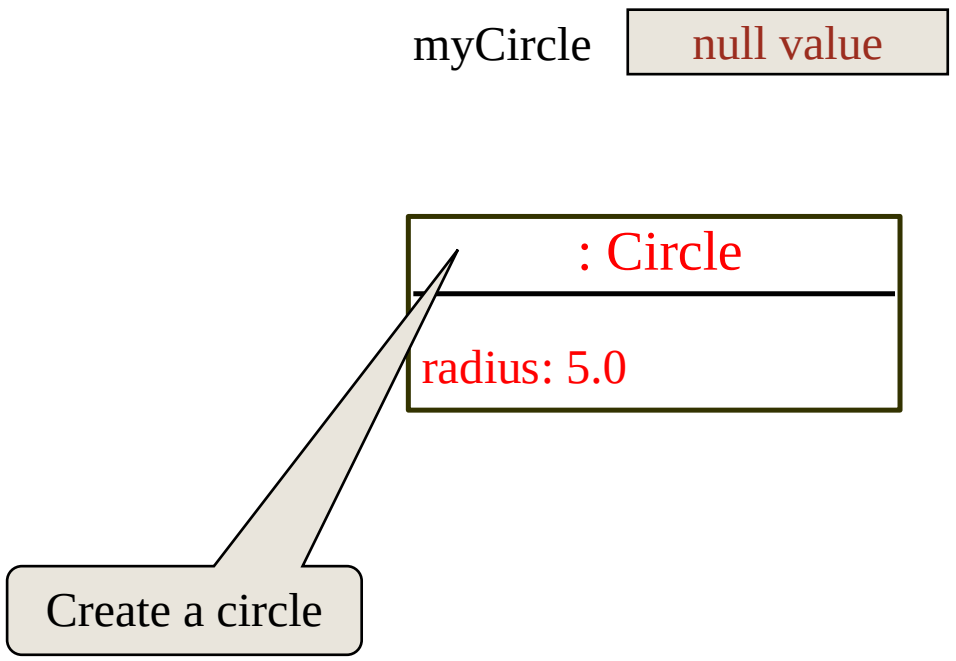
Using classes

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

SCircle yourCircle = new Circle();

yourCircle.radius = 100;
```



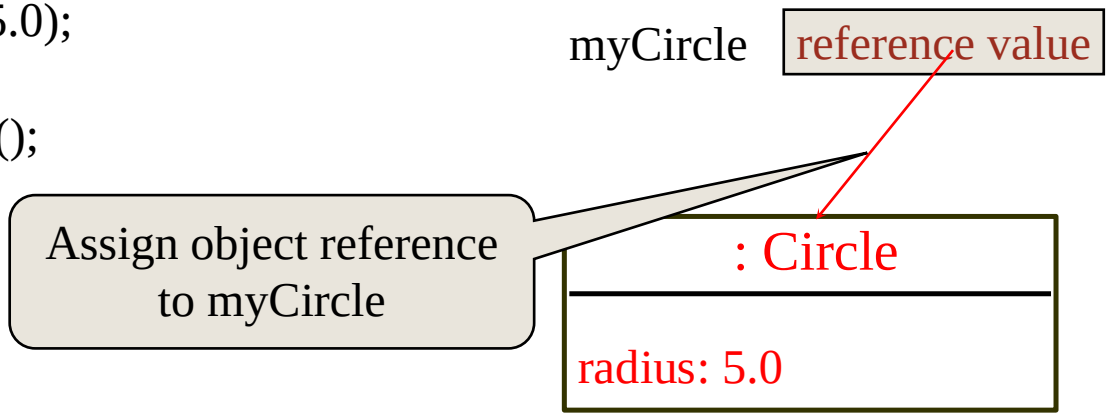
Using classes

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



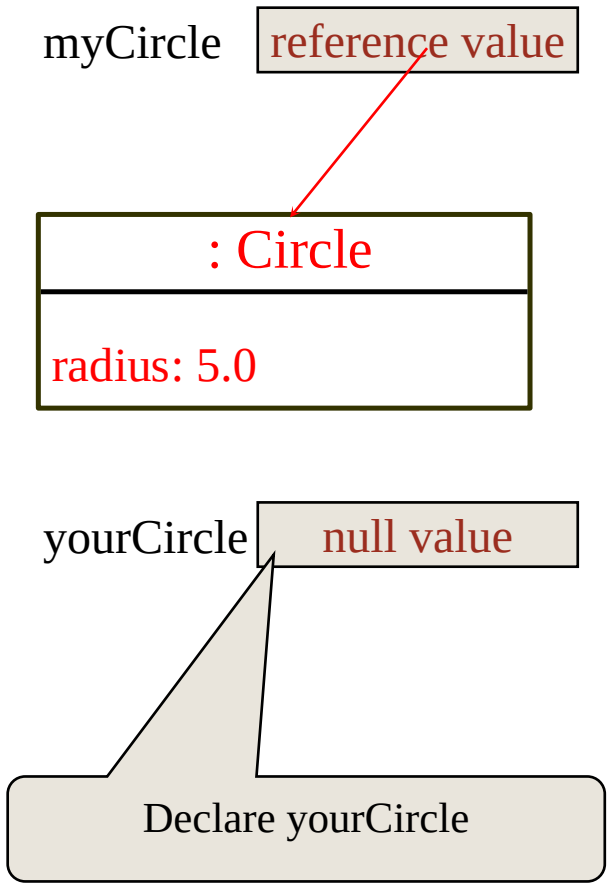
Using classes

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



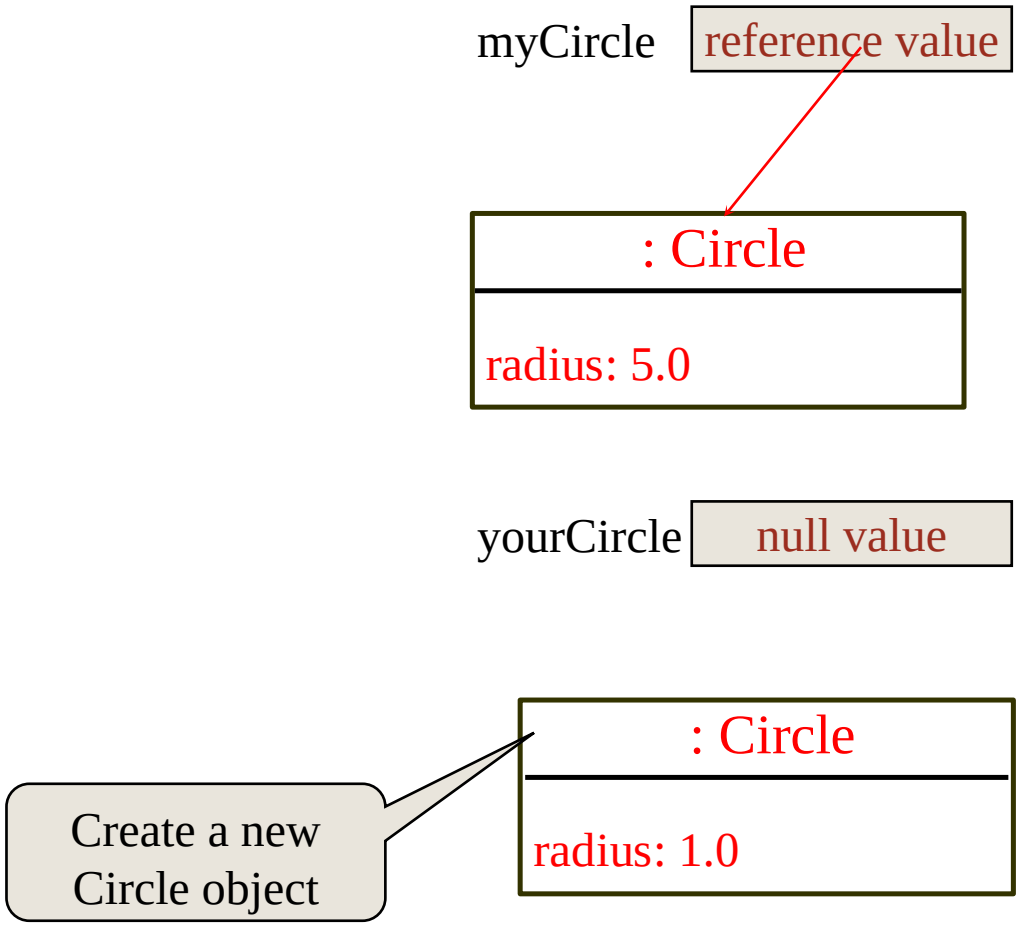
Using classes

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



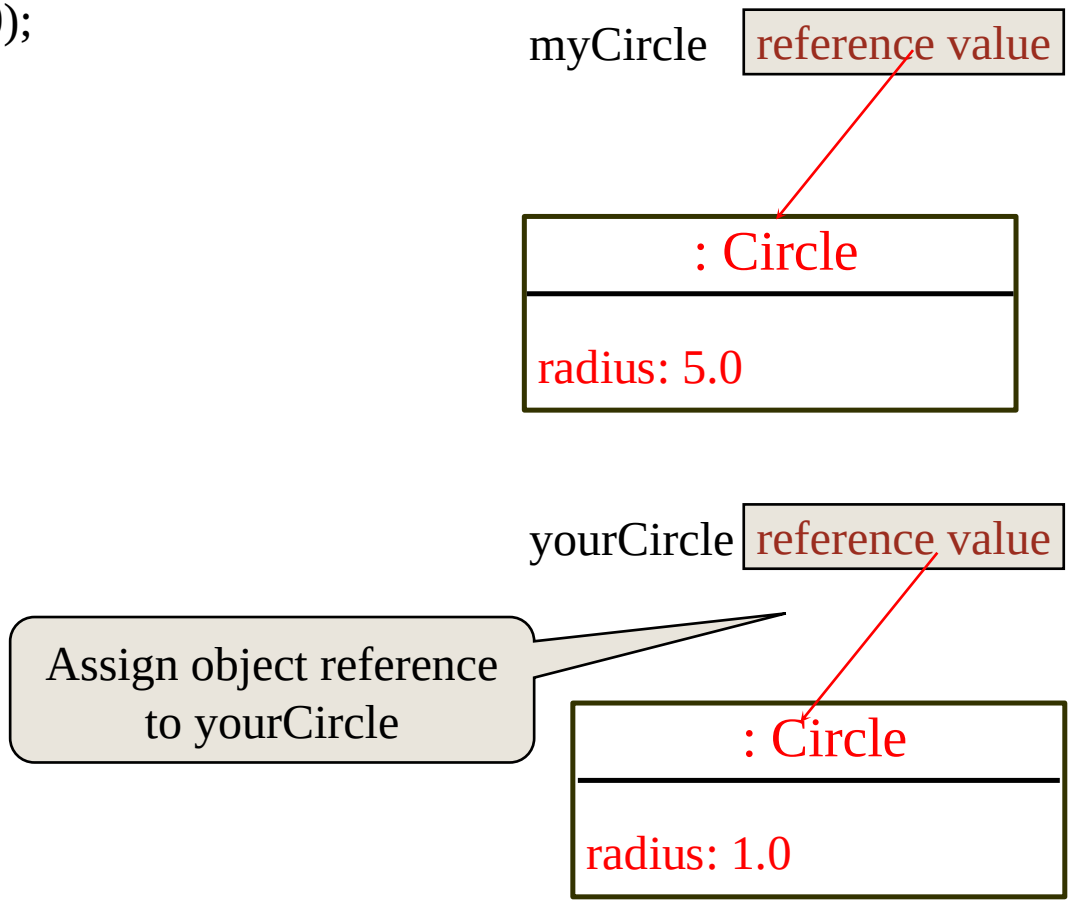
Using classes

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



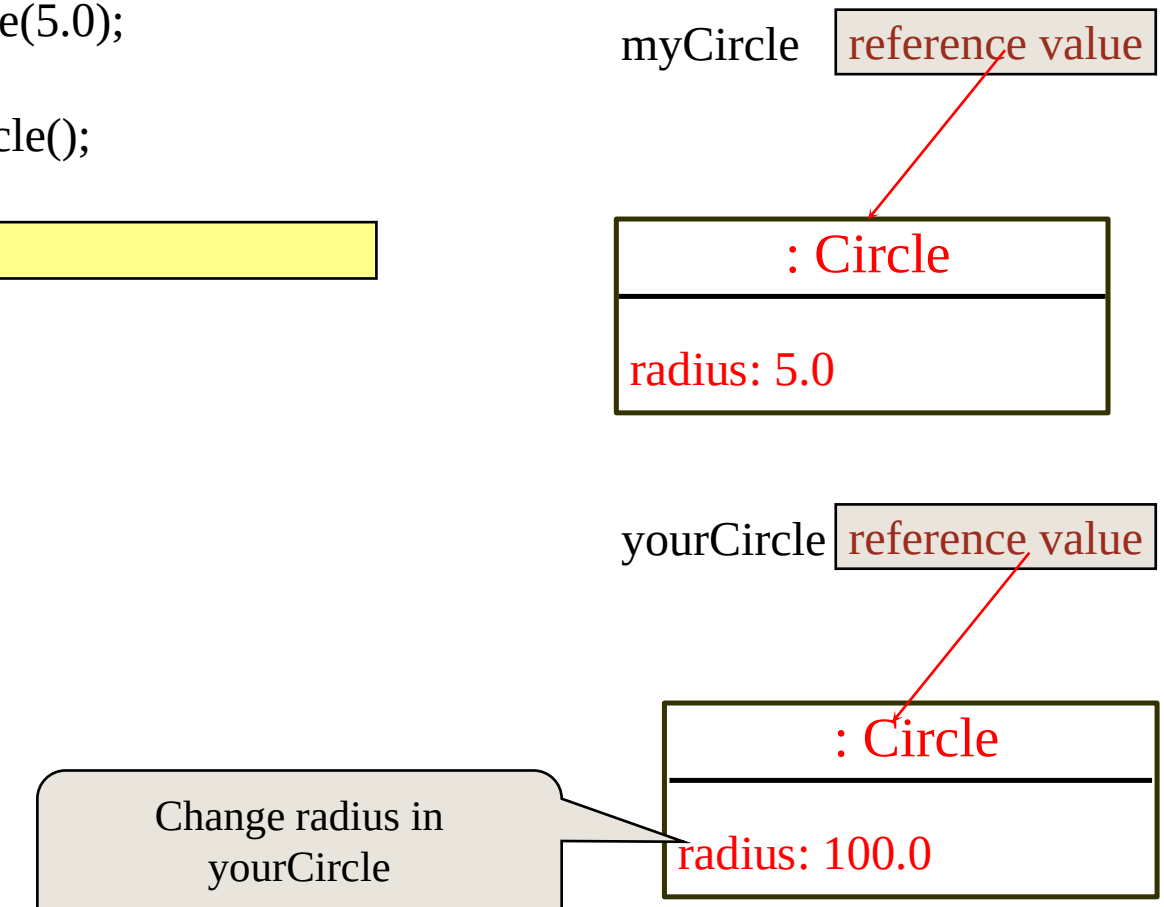
Using classes

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Using classes

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Static vs. Non-static variables

- Static variables and constants:
  - **global variables for the entire class: for all objects instances of this class**

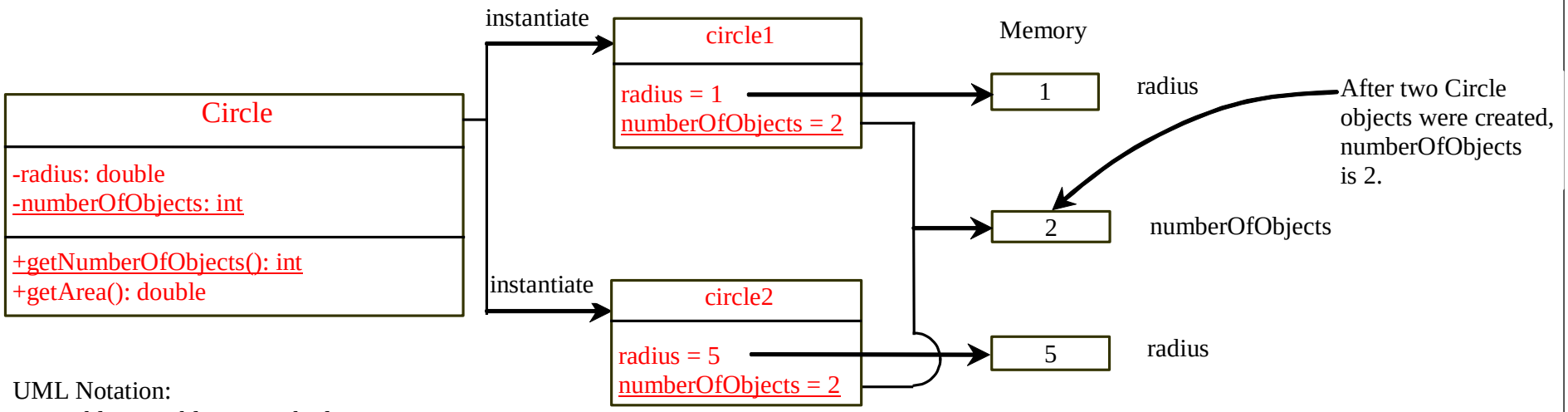
```
static int count = 0;

static final double PI = 3.141592;
```

- Non-static/instance variables are date fields of objects:

```
System.out.println(myCircle.radius);

System.out.println(yourCircle.radius);
```

Static Variables, Constants and Methods

- Static variables are shared by all the instances of the class:



Static vs. Non-static methods

- Static methods:
  - **Shared by all the instances of the class - not tied to a specific object:**

```
double d = Math.pow(3, 2);
```

- Non-static/instance methods must be invoked from an object instance of the class:

```
double d1 = myCircle.getArea();
double d2 = yourCircle.getArea();
```

In-Class Quiz 1: Static vs Non-Static

- What is the correct code to print the value of x ?

```
1  public class InClassQuiz {
2
3      int x = 5;
4      static int y = 10;
5
6      public static void main(String[] args) {
7
8          System.out.println( );
9
10     }
11 }
```

- x
- this.x
- InClassQuiz.x
- new InClassQuiz().x

In-Class Quiz 2: Static vs Non-Static

- What is the correct code to print the value of y ?

```
1  public class InClassQuiz {
2
3      int x = 5;
4      static int y = 10;
5
6      public static void main(String[] args) {
7
8          System.out.println( );
9
10     }
11 }
```

- y
- this.y
- InClassQuiz.y
- new InClassQuiz().y

No Default values for local variables

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 errors: the variables are not initialized

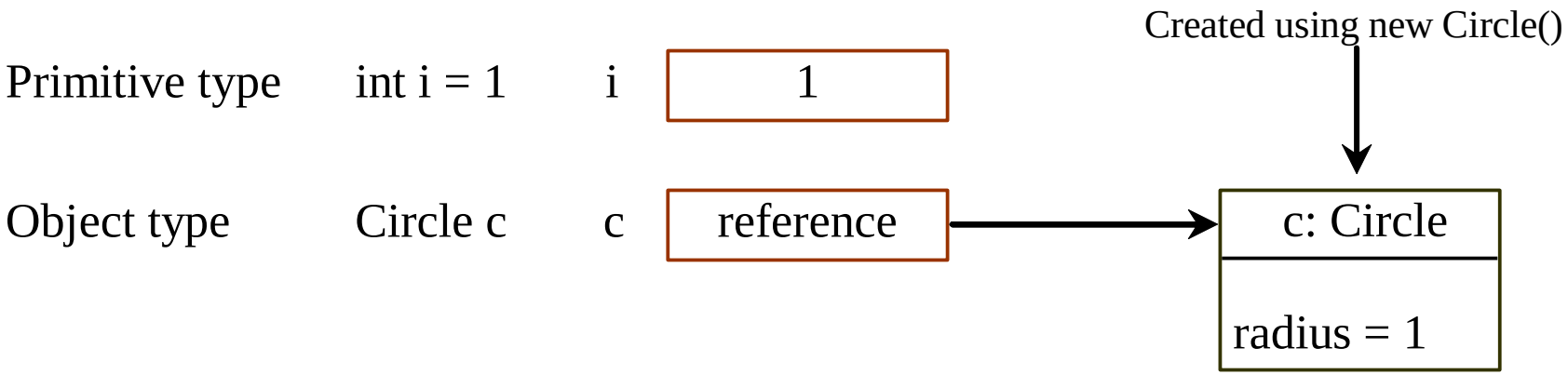
Default values for Data Fields

- Data fields have default values
  - Example:

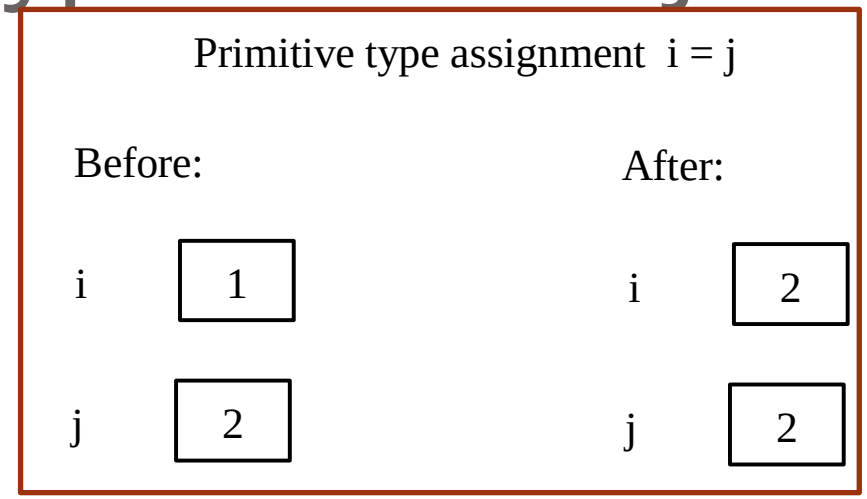
```
public class Student {
  String name; // name has default value null
  int age; // age has default value 0
  boolean isScienceMajor; // isScienceMajor has default value false
  char gender; // c has default value '\u0000'
}

public class Test {
  public static void main(String[] args) {
    Student student = new Student();
    System.out.println("name? " + student.name);              // null
    System.out.println("age? " + student.age);                // 0
    System.out.println("isScienceMajor? " + student.isScienceMajor);
                                                              // false
    System.out.println("gender? " + student.gender);          //
  }
}
```

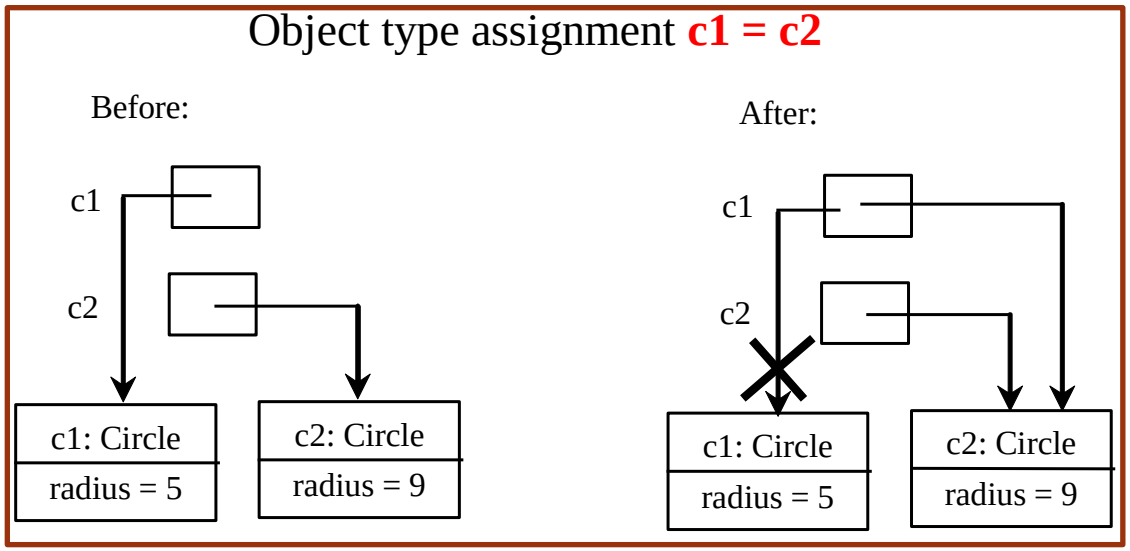
Note: If a data field of a reference type does not reference any object,the data field holds a special literal value: **null**.

Differences between Variables of Primitive Data Types and Object Types



Copying Variables of Primitive Data Types and Object Types





Garbage Collection

- The object previously referenced by c1 is no longer referenced, it is called ***garbage***
  - Garbage is automatically collected by the JVM, a process called ***garbage collection***
    - In older languages, like C and C++, one had to explicitly deallocate/delete unused data/objects

In-Class Quiz 3: Copying Object

- What is the output of the following code?

```
1  public class Example {
2
3      public int x;
4
5      public static void main(String[] args) {
6          Example obj1 = new Example();
7          obj1.x = 5;
8          Example obj2 = obj1;
9          obj2.x = 10;
10         System.out.println(obj1.x);
11     }
12
13 }
```

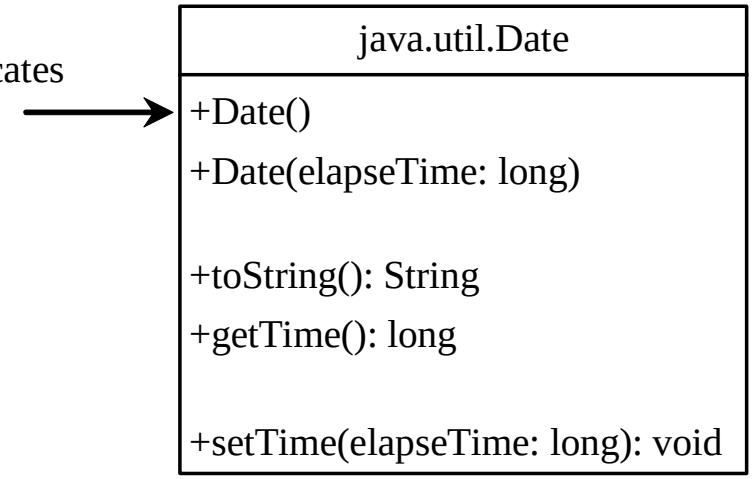
- x
- obj1.x
- 5
- 10

Example classes in Java: the Date class

Java provides a system-independent encapsulation of date and time in the java.util.Date class.

The toString method returns the date and time as a string

The + sign indicates public modifer



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.

January 1, 1970, GMT is called the Unix time (or Unix epoch time)

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

The Random class

java.util.Random

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

734 660 210 581 128 202 549 564 459 961

Visibility Modifiers and Accessor/Mutator Methods

- By default, the class, variable, or method can be accessed by any class in the same package.
  - `public` (+ in UML)

    The class, data, or method is visible to any class in any package.

  - `private` (- in UML)

    The data or methods can be accessed only by the declaring class - To protect data!

    - getField (accessors) and setField (mutators) methods are used to read and modify private properties.

Packages and modifiers

- The private modifier restricts access to within a class
- The default modifier restricts access to within a package
- public – unrestricted access

package p1;

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

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

```
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() {
    p1.C1 o = new p1.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();
  }
}
```

package p1;

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

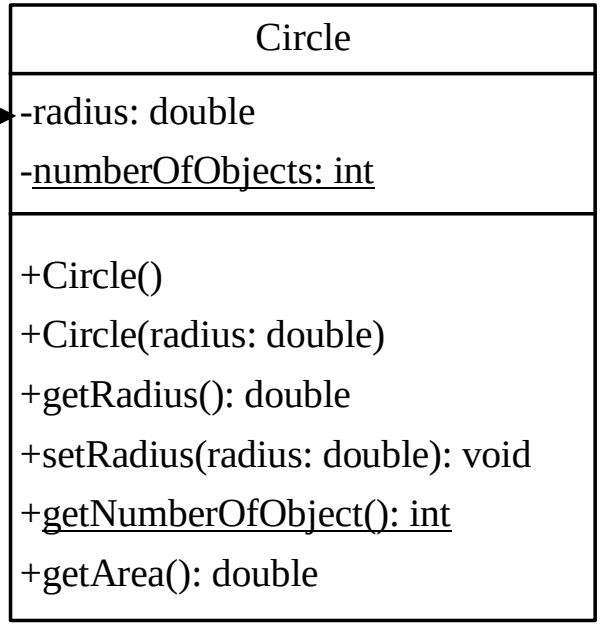
```
public class C2 {
  can access C1
}
```

package p2;

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

UML: Data Field Encapsulation

The - sign indicates private modifier



The radius of this circle (default: 1.0).

The number of circle objects created.

Constructs a default circle object.

Constructs a circle object with the specified radius.

Returns the radius of this circle.

Sets a new radius for this circle.

Returns the number of circle objects created.

Returns the area of this circle.

Array of Objects

- An array of objects is an *array of reference variables* (like the multi-dimensional arrays seen before)

```
Circle[] circleArray = new Circle[10];
circleArray[0] = new Circle();
circleArray[1] = new Circle(5);
```

...