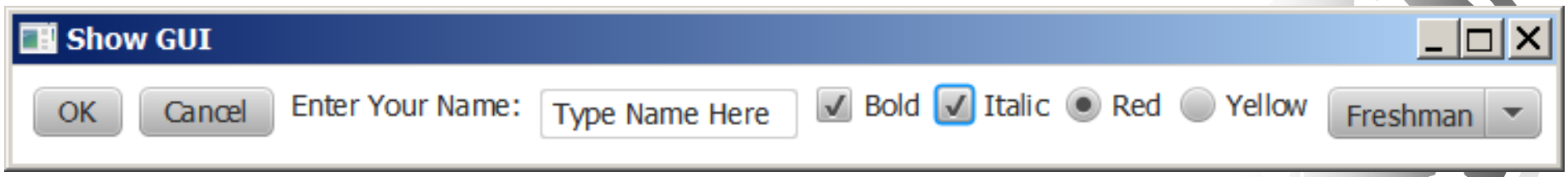
Chapter 9 Objects and Classes

Motivations

After learning the preceding chapters, you are capable of solving many programming problems using selections, loops, methods, and arrays.

However, these Java features are not sufficient for developing large scale software systems (and GUI).

Suppose you want to develop a graphical user interface as shown below. How do you program it?

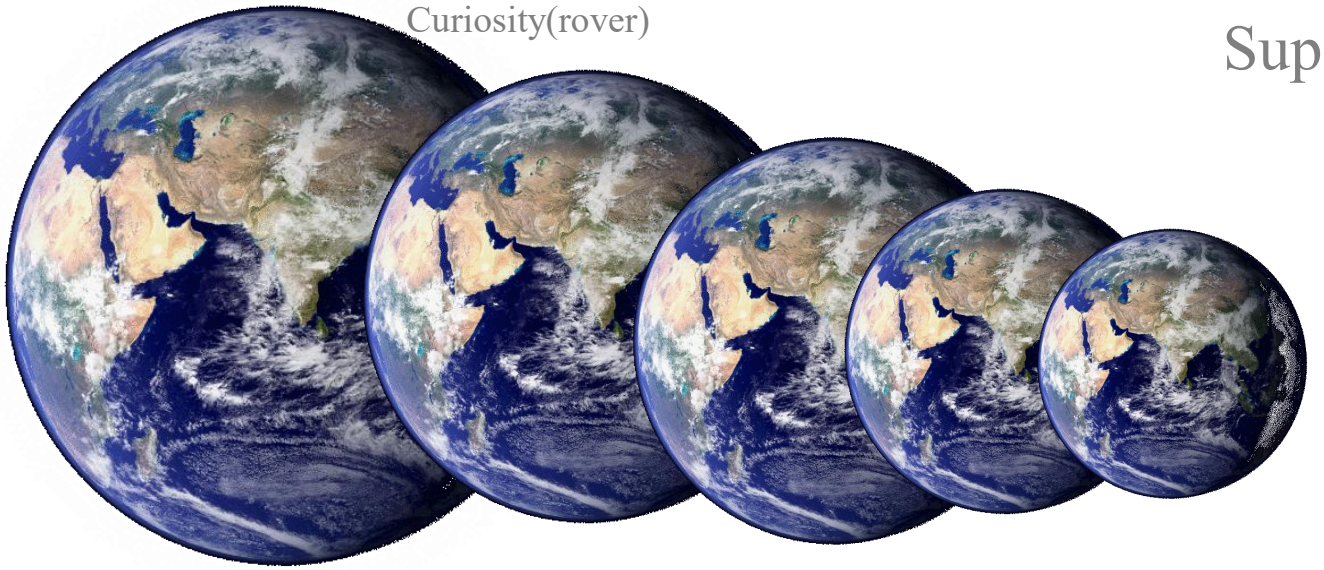


OUR TASK: creating software..

Complex Real World Tasks

Computer Assisted surgery

Curiosity(rover)



Super Tool (dumb?)

Two States - [0/1]- ??

electron

.. that runs on a computer to do some real world task..

From Real World Tasks to Objects

Complex Real World Tasks

System Analysis

System Design

Software Requirement Engineering

Software Design

- Classes
- Object Interactions
- UI
- ERD & DB
- etc…

From bits to Objects

- Bits

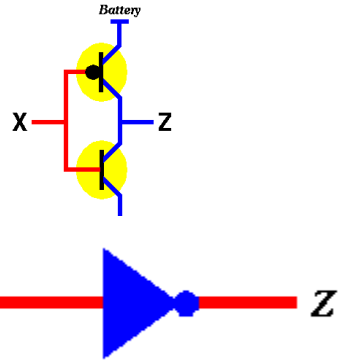
  Truth-tables

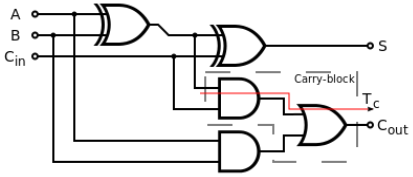
- Transistors

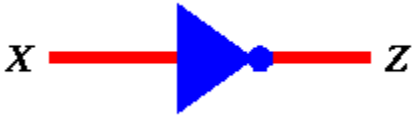
  Gates

- Adders, etc…

NOT	
0	?
1	?







All leads to **Machine Language**…

Examples: MIPS, ARM, x86, ….

From bits to Objects

From

- Machine Language (001000 01000 00000 0000000000000101)
- Assembly Language (addi $8, $0, 5)
- High level Languages (x = 5;)
- Object Oriented Languages
- Components
- Frameworks

High-Level Languages

- Operations:

```
totalCost = saleCost  + tax;
```

- Data Abstractions:

```
int x = 1; // 4 bytes
System.out.println(Integer.MAX_VALUE);  // 2147483647
System.out.println(Integer.MAX_VALUE + 1);  ???
```

- Control Abstractions (e.g. to beep 10 times)

```
for (int i=0; i <10; i++) { beep; }
```

- Modular Abstraction (e.g. to create a square function):

```
int square(int x) { return x * x; }
System.out.println(square(5));
```

But, need translators (???, ???)

OO Languages

All advantages of Structured Languages plus

- Encapsulation
- Inheritance
- Polymorphism

CLASSES, OBJECTS, INTERACTIONS

Objectives

- To describe objects and classes, and use classes to model objects (§9.2).
- To use UML graphical notation to describe classes and objects (§9.2).
- To demonstrate how to define classes and create objects (§9.3).
- To create objects using constructors (§9.4).
- To access objects via object reference variables (§9.5).
- To define a reference variable using a reference type (§9.5.1).
- To access an object's data and methods using the object member access operator (**.**) (§9.5.2).
- To define data fields of reference types and assign default values for an object's data fields (§9.5.3).
- To distinguish between object reference variables and primitive data type variables (§9.5.4).
- To use the Java library classes **Date**, **Random**, and **Point2D** (§9.6).
- To distinguish between instance and static variables and methods (§9.7).
- To define private data fields with appropriate **get** and **set** methods (§9.8).
- To encapsulate data fields to make classes easy to maintain (§9.9).
- To develop methods with object arguments and differentiate between primitive-type arguments and object-type arguments (§9.10).
- To store and process objects in arrays (§9.11).
- To create immutable objects from immutable classes to protect the contents of objects (§9.12).
- To determine the scope of variables in the context of a class (§9.13).
- To use the keyword **this** to refer to the calling object itself (§9.14).

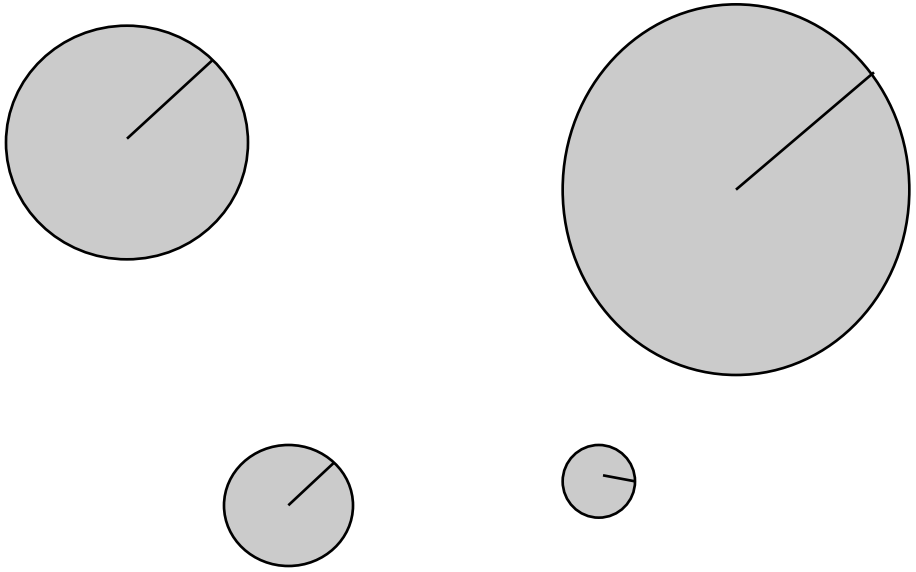
OO Programming Concepts

- OOP involves programming using objects.
- An *object* represents an entity in the real world that can be distinctly identified.

  E.g. a student, a loan, a circle, a button,
- An object has a unique identity, state, and behaviors.
- The *state* of an object consists of a set of *data fields* (also known as *properties*) with their current values.
- The *behavior* of an object is defined by a set of methods.

Circle Objects

All have a common property, radius (with different values).



For all of these objects area can be calculated using the same method (area = 2 * PI * radius).
All are CIRCLES.

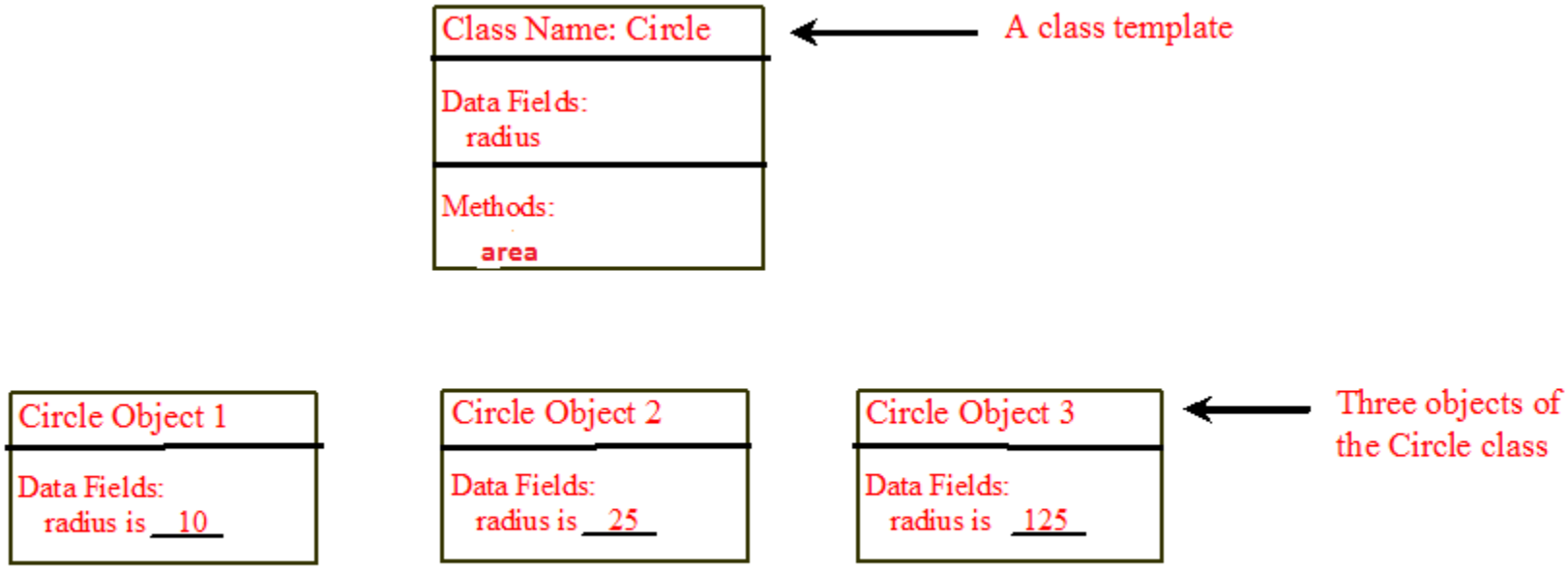
Classes

Classes are templates that define objects of the same type.

A Java class uses variables to define data fields and methods to define behaviors.

Additionally, a class provides a special type of methods, known as constructors,
used to construct objects of that class.

Objects- Instances of Classes



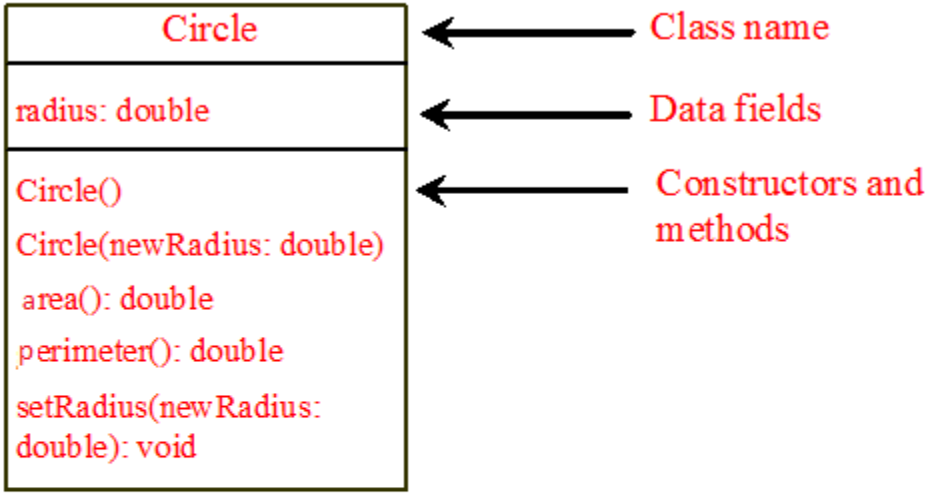
An object has both a state and behavior.

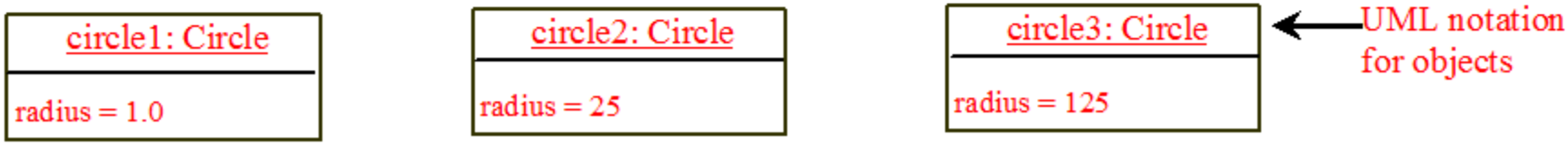
state defines the object,

behavior defines what the object does.

UML Class Diagram

UML Class Diagram





Example: Circle

- OPEN Eclipse & Create a Project (say Shape)
- Create a Package (say Shapes1)
- Create a Class named Circle in this package.
- Include code for Circle Class (next slide)
- Create a Class TestShape (on the same package)
- Include code for testCircle( ) method (slide 17) & call it in main()

Circle Class

```
public class Circle {

    private double radius;

    public Circle(double radi){
        radius = radi;
    }

    public double diameter(){
        return 2* radius;
    }

    public double perimeter(){
        return 2* Math.PI * radius;
    }
```

Data field

constructor

method

method

Note that the keyword static is not used

CLASS as a TYPE

- If compiled, it will create a user-defined type (e.g Circle).
  - A meta object will be created (Circle object – study later)

To use this user-defined type,

- We have to declare a variable of this type

```
Circle circle1;
```

- And then, create the variable using new (will use the constructor- see slides 26)

```
circle1 = new Circle(5.0);
```

OR combine both ; Circle circle1 = new Circle(5.0);

Test Class

```
public class TestShape {
public static void testCircle() {
    Scanner scan1= new Scanner(System.in);
    System.out.println("Give the radius of the circle:");
    double radius = scan1.nextDouble();
    Circle circle1 = new Circle(radius); //constructs circle1
    System.out.printf("The diameter of the circle is: %.2f
       %n" , circle1.diameter());
    System.out.printf("The perimeter of the circle is: %.2f
       %n", circle1.perimeter());
    // can we print circle1.area()?
    scan1.close();
   }
  public static void main(String[] args) {
       testCircle();
}
```

some more methods to Circle

- To return area of the circle
- To check whether another circle is equal to the current circle (this)
  - public boolean equal(Circle c)
- To check whether another circle is larger than the current circle (this)
  - public boolean larger(Circle c)

SET and GET methods

- Recommended to declare Data Fields as PRIVATE (why ? – later)
- PRIVATE members can be accessed only by the methods in the same class.
- Therefore, you have to provide PUBLIC Methods to access these PRIVATE Data Fields
- Example: private radius;

Circle (cont.)

```
public int getRadius(){
        return radius;
}

public void setRadius(int radi){
        radius= radi;
}
public double area(){
        return Math.PI * radius  * Radius;
}
public boolean isEqualTo(Circle C){
        return (this.radius ==C.radius);
}
public boolean larger(Circle C){
        return (this.radius < C.radius);
}
```

Test Class (cont.)

```
public class Test {

  public static void testCircle() {
        //continue from previous
      System.out.printf("The area of the circle is: %.2f %n" ,
      circle1.area());
      System.out.println("Give the radius of another circle:"),
      radius = scan1.nextDouble();
      Circle circle2 = new Circle(radius);
      if (circle1.isEqualTo(circle2))  // or circle2.isEqualTo(circle1)
            System.out.println("Circles are equal");
      else
            System.out.println("Circles are NOT equal");
   //NOTE: check circle1.equals(circle2) ??
  }
  public static void main(String[] args) {
      testCircle();
}
```

Rectangle Class

- Write Rectangular class
- Include suitable data fields (?, ?)
- Include area(), perimeter(), etc..
- Write a method (say testRectangular()) in the Test class to test it.

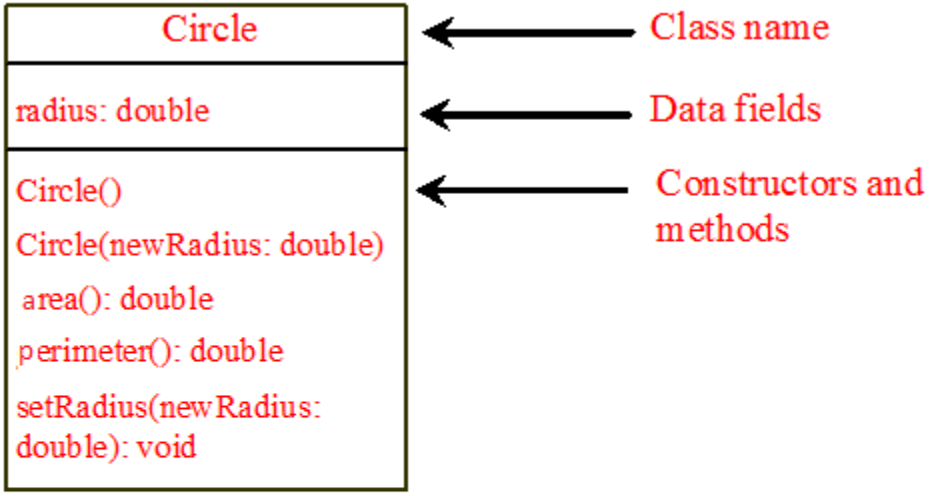
Triangle Class

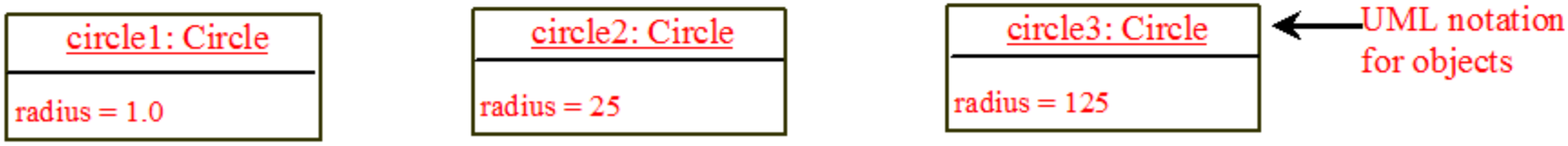
- Write Triangular class
- Include suitable data fields (??)
- Include area(), perimeter(), etc..
- Write a method testTriangular() in the Test class.

UML
Class Diagram

UML Class Diagram

UML Class Diagram





Example: Defining Classes and Creating Objects

TV	
`channel: int`	The current channel (1 to 120) of this TV.
`volumeLevel: int`	The current volume level (1 to 7) of this TV.
`on: boolean`	Indicates whether this TV is on/off.
`+TV()`	Constructs a default TV object.
`+turnOn(): void`	Turns on this TV.
`+turnOff(): void`	Turns off this TV.
`+setChannel(newChannel: int): void`	Sets a new channel for this TV.
`+setVolume(newVolumeLevel: int): void`	Sets a new volume level for this TV.
`+channelUp(): void`	Increases the channel number by 1.
`+channelDown(): void`	Decreases the channel number by 1.
`+volumeUp(): void`	Increases the volume level by 1.
`+volumeDown(): void`	Decreases the volume level by 1.

The + sign indicates a public modifier.

TV

TestTV

Run

CONSTRUCTORS

Constructors

Constructors are a special kind of methods that are invoked to construct objects.

```
private double radius;

//construct a circle with default radius 0.0
public Circle() {
}

//construct a circle with given radius
public Circle(double newRadius) {
  radius = newRadius;
}
```

Constructors, cont.

Constructors must have the same name as the class itself.

Constructors do not have a return type—not even void.

Constructors are invoked using the new operator when an object is created.

Constructors play the role of initializing objects.

Constructors are usually overloaded (?)

Default Constructor

- A constructor with no parameters is referred to as a *no-arg constructor*.
- 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 is called *a default constructor*
  - provided automatically ***only if*** *no constructors are explicitly defined in the class*.
- If you include your own constructor, JAVA assumes, you do not want to include no-arg empty-body constructor [if you need this, you have to explicitly include it]

Creating Objects Using Constructors

```
new ClassName();
```

Example:

```
new Circle();

new Circle(5.0);
```

Check- which pair will not compile?

```
public class Circle {
      private double radius;
   // NO EXPLICIT CONSTRUCTOR
}
```

```
Circle circle1 = new Circle();
circle1.setRadius(radius);
```

```
public class Circle {
      private double radius;
      public Circle(double radi){
            radius=radi;
      }
}
```

```
Circle circle1 = new Circle();
circle1.setRadius(radius);
```

Check- which pair will not compile?

```
public class Circle {
      private double radius;
      public Circle(double radi){
            radius=radi;
      }
      public Circle(){
      }
}
```

```
Circle circle1 = new Circle();
circle1.setRadius(radius);
```

CONSTRUCTING Objects

```
public class Circle {
        private double radius;
        public Circle(double radi){
                radius=radi;
        }
        //default circle with radius 1.0
        public Circle(){
                radius = 1.0
        }
}
```

```
public class Circle {
        private double radius = 1.0;
        public Circle(double radi){
                radius=radi;
        }
        //default circle with radius 1.0
        public Circle(){
        }

}
```

```
public class Circle {
        private double radius;
        public Circle(double radi){
                radius=radi;
        }
        // circle with default radius 0.0
        public Circle(){
        }

}
```

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();
```

Assign object reference

Create an object

Example:

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

```
public class Circle {
        private double radius = 1.0; // if not initialized, java will assign 0.0
        public Circle(double radi){
                radius=radi;
        }
        public Circle(){
        }
}
```

animation

Trace Code

Declare myCircle

myCircle | no value

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

```
public class Circle {
        private double radius = 1.0; // if not initialized, java will assign 0.0
        public Circle(double radi){
                radius=radi;
        }
        public Circle(){
        }
}
```

animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

myCircle | no value

: Circle

radius: 5.0

```
public class Circle {
        private double radius = 1.0;
        public Circle(double radi){
                radius=radi;
        }
        public Circle(){
        }
}
```

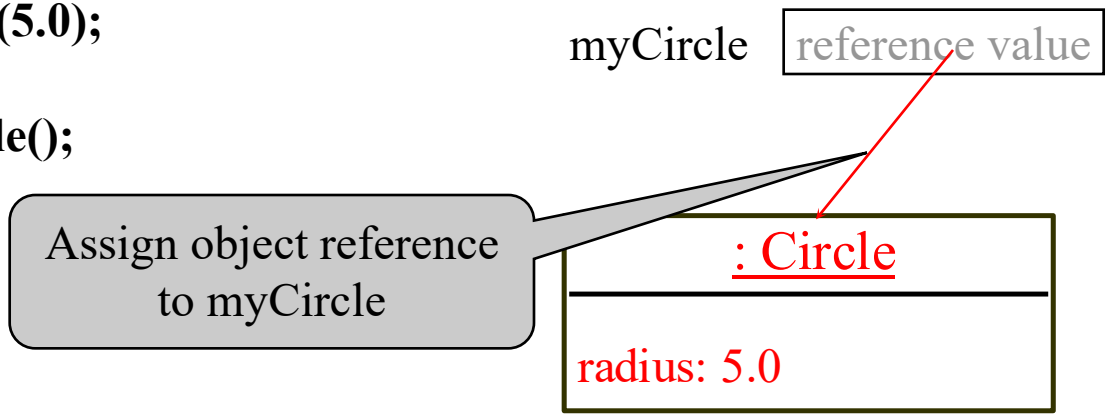
Create a circle

animation

Trace Code, cont.

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();




```
public class Circle {
        private double radius = 1.0;
        public Circle(double radi){
                radius=radi;
        }
        public Circle(){
        }
}
```

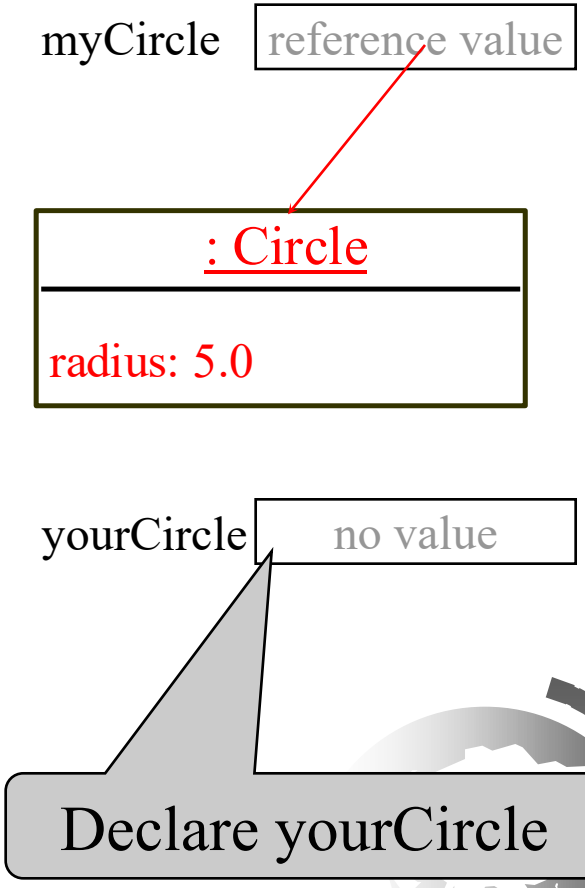
animation

Trace Code, cont.

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

```
public class Circle {
        private double radius = 1.0;
        public Circle(double radi){
                radius=radi;
        }
        public Circle(){
        }
}
```



animation

Trace Code, cont.

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

```
public class Circle {
        private double radius = 1.0;
        public Circle(double radi){
                radius=radi;
        }
        public Circle(){
        }
}
```

myCircle reference value

: Circle

radius: 5.0

yourCircle no value

: Circle

radius: 1.0

Create a new Circle object

animation

Trace Code, cont.

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

```
public class Circle {
        private double radius = 1.0;
        public Circle(double radi){
                radius=radi;
        }
        public Circle(){
        }
}
```

myCircle **reference value**

: Circle

radius: 5.0

yourCircle **reference value**

Assign object reference to yourCircle

: Circle

radius: 1.0

Accessing Object's Members

- Referencing the object's data directly: // NOT GOOD

  `objectRefVar.data //` if data is private

  e.g., `myCircle.radius //` if radius is public

  GOOD practice is: declare radius as private,

  and provide public methods to SET and GET radius

- Invoking the object's method:

  `objectRefVar.methodName(arguments)`

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

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

animation

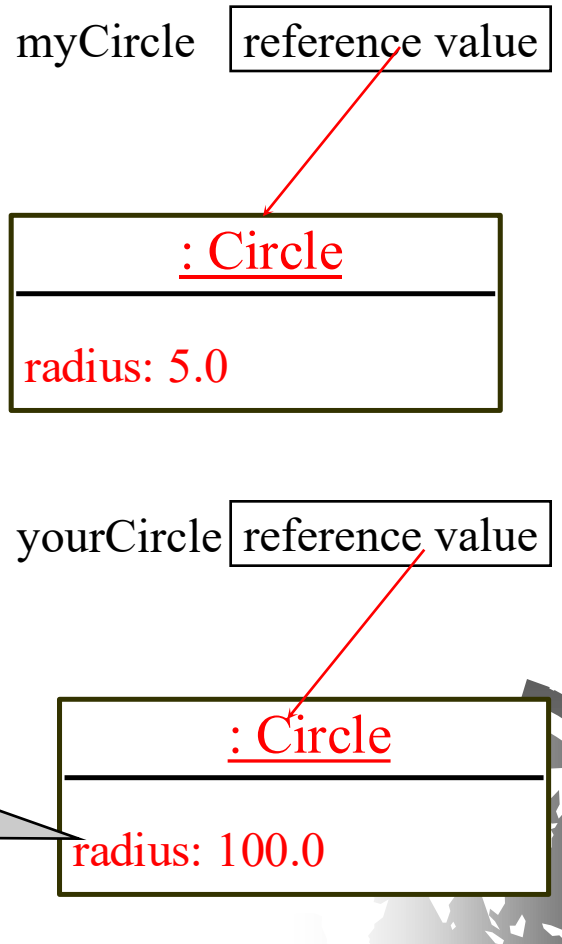
Trace Code, cont.

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.setRadius(100.0);

```
public void setRadius(int radi){
                radius= radi;
}
```



Default Values
Data Fields (declared in class outside methods)
Vs
Local Variables (declared inside methods)

Default Values for Data Fields

If a data field is not initialized, Java will initialize them with default values.

- null for a reference type,
  - e.g. int[] A; Circle circle1;
- 0 for a numeric type,
  - e.g. int A; double B;
- false for a boolean type,
  - e.g. boolean A;
- '\u0000' for a char type.
  - char A;

The null Value

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

For example, consider the following Student class. It contains three Primitive Data Fields and two Reference Data Fields.

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

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

 public static void main(String[] args) {
        Student1 s1 = new Student1();
        System.out.printf("%s %s %d %b %c", s1.name, s1.dateOFbirth,
               s1.age, s1.isLocal,s1.gender);
 }
```

null null 0 false

But, what if we declare a local variable without Any initial value in main() and try to print it?

No Default Values for Local Variables

Java will not assign a 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

Comparing and Copying

Primitive Data Types
Vs
Reference Types (e.g. Objects)

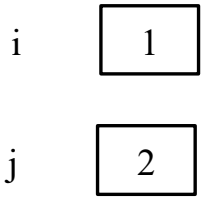
Comparing Objects
What will be printed?

```
Circle circle1 = new Circle(5.0);
Circle circle2 = new Circle(5.0);
System.out.printf( "%b %b %b",
                   (circle1 == circle2),
                   circle1.equal(circle2), //java supplies this
                   circle.isEqualTo(circle2)); // our method
// false false true
circle2 = circle1;  // old circle2 will become garbage in heap;
System.out.printf("%b %b %b",
                   (circle1 == circle2),
                    circle1.equals(circle2),
                   circle1.isEqualTo(circle2));
// true true true
```

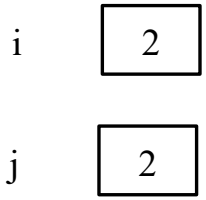
Copying Variables of Primitive Data Types and Object Types

Primitive type assignment i = j

Before:

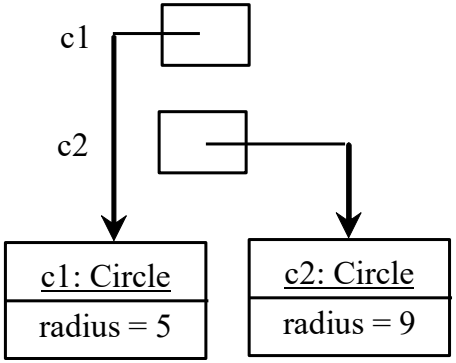


After:

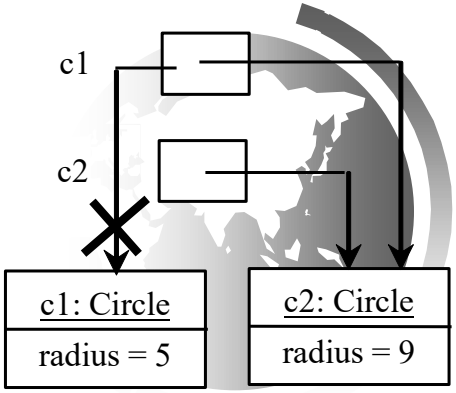


Object type assignment c1 = c2

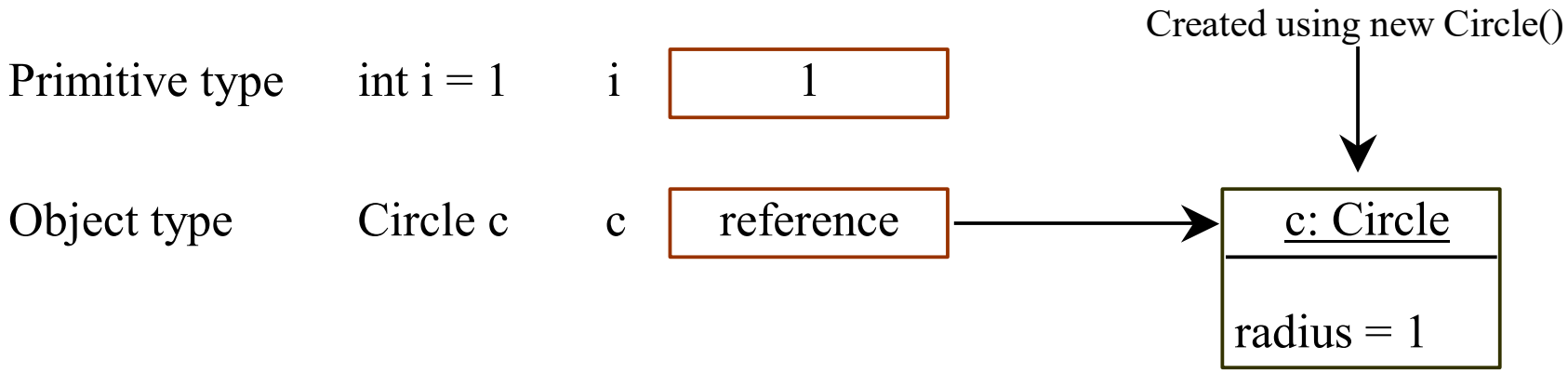
Before:



After:



Differences between Variables of Primitive Data Types and Object Types



What will be printed?

```
int i = 1;
int j = i;
i = 3;
print j
```

```
Circle c1 = new Circle(5.0);
Circle c2 = new Circle(4.0);
c2 = c1; // old c2 is garbage
c1. setRadius(3.0)
print c2.radius();
```

Garbage & Garbage Collection

See previous slide,

after c1 = c2,

c1 and c2 points to the same object.

The object previously referenced by c1 is no longer referenced.

This object is known as garbage.

Garbage is automatically collected by JVM.

TIP: Garbage Collection

If 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 .

```
Circle c1 = new Circle(5.0);
c1 = null // old c1 is garbage
```

CLASSES
in Java Built-in Libraries
examples:
Date in java.util
Point2D in javafx.geometry
Random in java.util

[Math.random() is limited – it is a static method in Math class- just returns a random number x, $0.0 \leq x < 1.0$]

The Date Class

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

The + sign indicates public modifer

java.util.Date	
+Date()	Constructs a Date object for the current time.
+Date(elapseTime: long)	Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.
+toString(): String	Returns a string representing the date and time.
+getTime(): long	Returns the number of milliseconds since January 1, 1970, GMT.
+setTime(elapseTime: long): void	Sets a new elapse time in the object.

The Date Class Example

For example, the following code

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

displays a string like `Sun Mar 09 13:50:19 EST 2003`.

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.

The 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

The **Point2D** Class

Java API has a conveninent **Point2D** class in the **javafx.geometry** package for representing a point in a two-dimensional plane.

javafx.geometry.Point2D	
`+Point2D(x: double, y: double)`	Constructs a `Point2D` object with the specified *x*- and *y*-coordinates.
`+distance(x: double, y: double): double`	Returns the distance between this point and the specified point (*x*, *y*).
`+distance(p: Point2D): double`	Returns the distance between this point and the specified point p.
`+getX(): double`	Returns the *x*-coordinate from this point.
`+getY(): double`	Returns the *y*-coordinate from this point.
`+toString(): String`	Returns a string representation for the point.

TestPoint2D Run

CLASS (STATIC) members
Vs
OBJECT (INSTANCE) members

Static Variables, Constants, and Methods

- Static variables are shared by all the instances of the class.
  - e.g. private static int count; // counts number of objects
- Static methods are not tied to a specific object.
  - e.g. public static getCount(); // to return the count
- Static constants are final variables shared by all the instances of the class.
  - Suppose Math.PI is not available, in Circle class we should include: private static final double PI = 3.237

Instance Variables, and Methods

- Instance variables belong to a specific instance
  - e.g. private double radius;
- Instance methods are invoked by an instance of the class
  - e.g. public double area()
- Instance Constant is same as Class Constant(but wastes memory)
  - Use static followed by final instead of just final

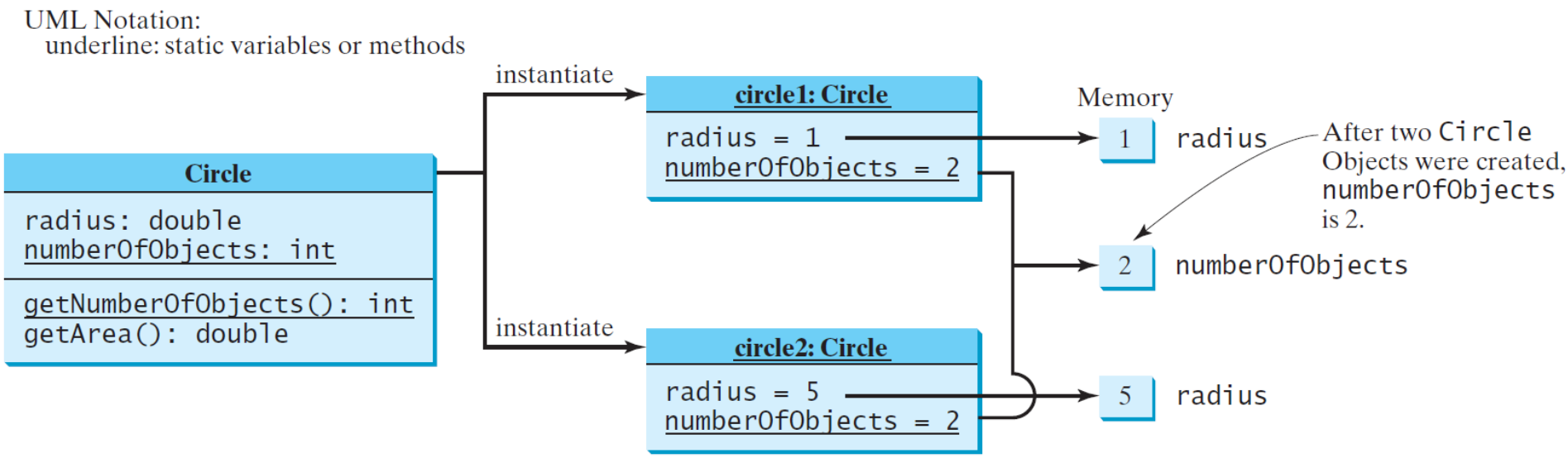
Examples: STATIC Constants, Variables & Methods

```java
package Shape;

public class Circle {
   private double radius;
   static final double PI = 3.27; // this.PI can be used in calculations
   private static int count = 0;
   public Circle(double radi){
       radius=radi;
       count++;
   }
   public Circle() {
       count++;
    }
   public static int count(){
       return count;
    }
```

```java
private static void testStatic(){
//java automatically creates a meta object Circle. No need to construct using new
    System.out.println("The PI value used in Circle class is " + Circle.PI); // note: new is not used
    System.out.println("The number of circles constructed is " + Circle.count());
    Circle c1 = new Circle(5.0);
    Circle c2 = new Circle(6.0);
   System.out.println("The number of circles constructed is " + Circle.count());
}
```

Static Variables, Constants, and Methods, cont.



Example of Using Instance and Class Variables and Method

Objective: Demonstrate the roles of instance and class variables and their uses. This example adds a class variable numberOfObjects to track the number of Circle objects created.

CircleWithStaticMembers

TestCircleWithStaticMembers

Run

VISIBILITY and ACCESS MODIFIERS

- PUBLIC
- PROTECTED
- PRIVATE
- What is Default?

ACCESS Modifiers

- `public`

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

- `private`

  The data or methods can be accessed only by the declaring class. The get and set methods are used to read and modify private properties.

DEFUALT : By default, the class, variable, or method can be accessed by any class in the same PACKEAGE. It is a limited PUBLIC

```java
package p1;

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

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

```java
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();
  }
}
```

```java
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();
  }
}
```

The private modifier restricts access to within a class, the default modifier restricts access to within a package, and the public modifier enables unrestricted access.

```java
package p1;

class C1 {
  ...
}
```

```java
package p1;

public class C2 {
  can access C1
}
```

```java
package p2;

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

The default modifier on a class restricts access to within a package, and the public modifier enables unrestricted access.

NOTE

An object cannot access its private members, if the object is used in an object of some other class as shown in (b).
It is OK, however, if the object is declared in its own class, as shown in (a).

```java
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**.

```java
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**.

ENCAPSULATION

- Data Fields & Methods Together
- PRIVATE Data Fields[also some PRIVATE Methods](hiding)
- PUBLIC methods (interface)

Why Data Fields Should Be private?

To protect data.

To make code easy to maintain.

Example of Data Field Encapsulation

The - sign indicates private modifier

Circle	
-radius: double	The radius of this circle (default: 1.0).
-numberOfObjects: int	The number of circle objects created.
+Circle()	Constructs a default circle object.
+Circle(radius: double)	Constructs a circle object with the specified radius.
+getRadius(): double	Returns the radius of this circle.
+setRadius(radius: double): void	Sets a new radius for this circle.
+getNumberOfObjects(): int	Returns the number of circle objects created.
+getArea(): double	Returns the area of this circle.

CircleWithPrivateDataFields

TestCircleWithPrivateDataFields

Run

OBJECTS as Method's Parameters

Java pass arguments by value only.

Pass By Value

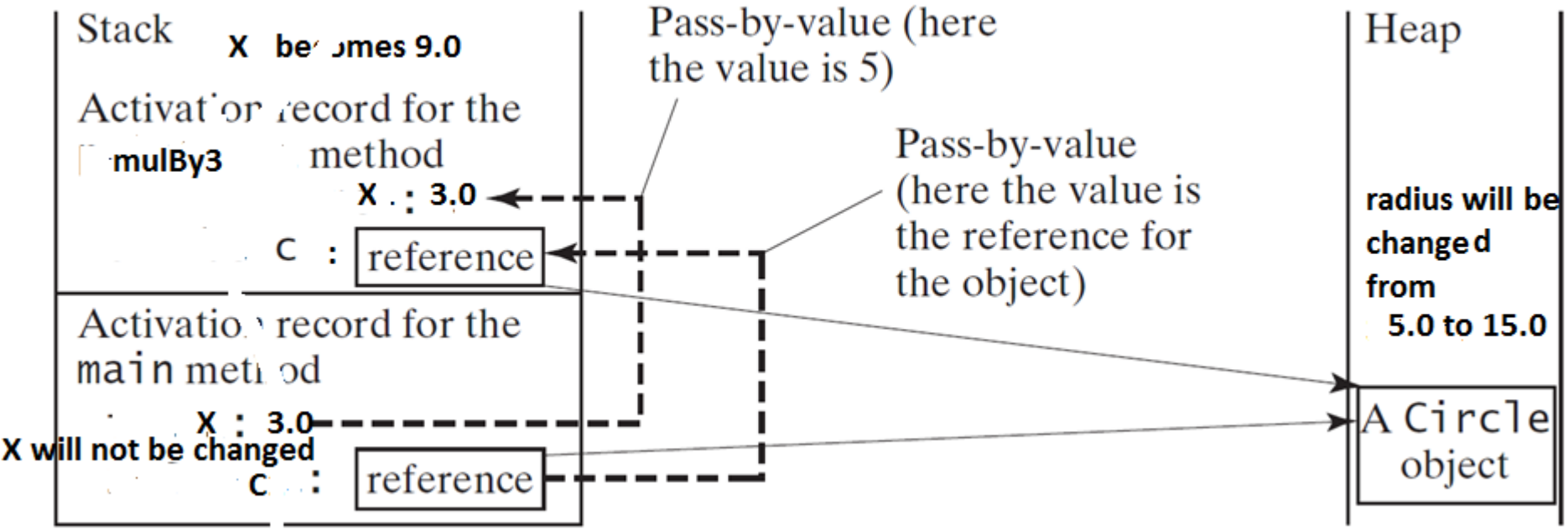
- Java pass arguments by value only
  - That is, a local copy is made in the stack area for the method
  - If a variable of primitive data type is passed there will be no side effect.
  - If a variable of reference type (e.g. an object) is passed, there may be side effects.

```
//include in Test
private static void mulBy3(double x, Circle c1){
    x = x * 3;
    c1.setRadius(x);
    }
public static void testPassByRef() {
    double x = 3.0;
    Circle c1 = new Circle(x);
     mulBy3(x, c1);
     System.out.printf("%f %f",  x , c1.getRadius());
  }
```

```
public static void main(String[] args) {
      //testCircle();
       testPassByRef();
}
```

```
3.000000 15.000000
```

Pass by Value
Primitive Vs Reference

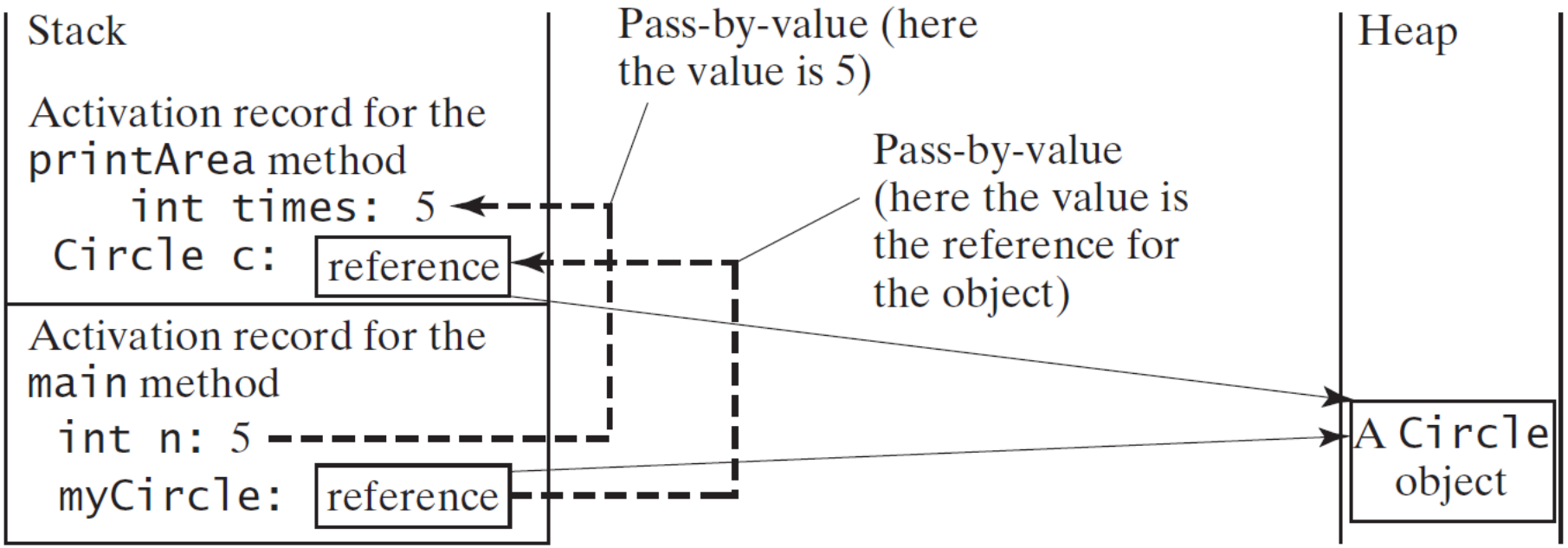


Passing Objects to Methods

- Passing by value for primitive type value (the value is passed to the parameter)
- Passing by value for reference type value (the value is the reference to the object)

TestPassObject Run

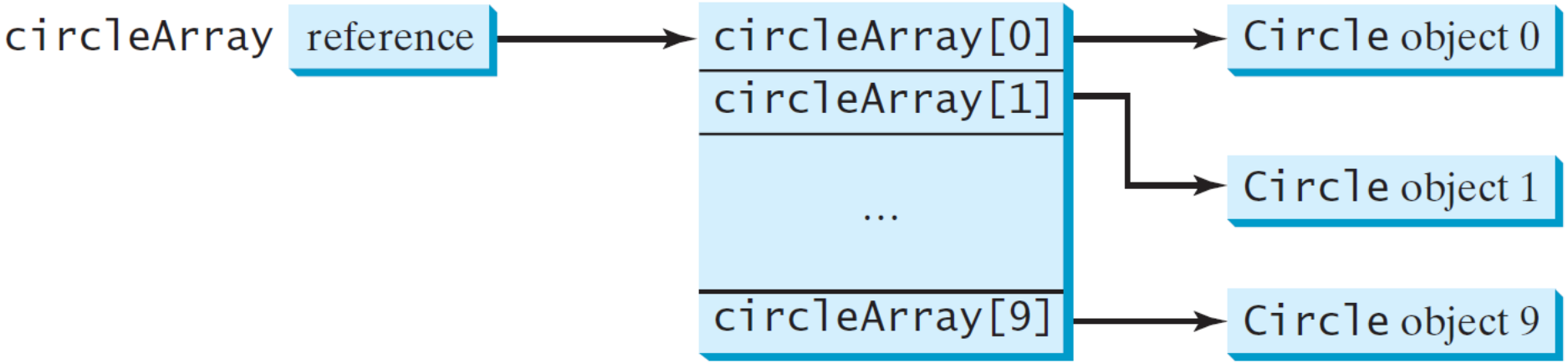
Passing Objects to Methods, cont.



Array of Objects

```
Circle[] circleArray = new Circle[10];
circleArray[0] = new Circle(5.0);
circleArray[1] = new Circle(3.0);
```

TWO LEVELS OF REFERENCING



Array of Objects

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

An array of objects is actually an *array of reference variables*. circleArray[1].getArea() involves two levels of referencing.

circleArray references to the entire array. circleArray[0] references to the first Circle object in the array.

Array of Objects- Example

Create 10 circles with random radius and calculate the total perimeter.

```
public void testArray(){
   Circle[] circleArray = new Circle[10];
   for (int i = 0; i <10 ; i++){
        double radi = Math.random()*10;
        circleArray[i] = new Circle(radi);
   }
   double totalPerimeter=0;
   for (Circle circle:circleArray)
      totalPerimeter+= circle.perimeter();
      System.out.println("Total perimeter is :“
          + totalPerimeter);
   }
   }
```

Array of Objects- Example

TotalArea

Run

Mutable
Vs
Immutable

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 set method in the Circle class the class would be immutable because radius is private and cannot be changed without a set method.

A class with all private data fields and without mutators is not necessarily immutable. For example, the Student class (next slide) has all private data fields and no mutators, but it is mutable.

Immutable objects/classes

- Only constructors modify the data fields.
- Once constructed its state (data fields) cannot be changed
- All data fields should be private
- No mutator methods included (e.g. setRadius())
- A data field may be a reference to an object- if so, that object should be mutable

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); // Now the student birth year is changed!
  }
}
```

What Class is Immutable?

For a class to be immutable, it must mark all data fields private and provide no mutator methods and no accessor methods that would return a reference to a mutable data field object.

Student class is not mutable, because it includes a method getBirthDate() that returns the reference to a mutable data field object birthDate. The object birthdate is mutable because one of its data field year can be modified by the method setYear()

SCOPE of the Variables

Scope of Variables

- The scope of data field (instance and static variables) is the entire class. They can be declared anywhere inside a class.
  - Java provides a default initial value
- 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.

this

refers to the current object

```
public boolean isEqualTo(Circle c){
            return (this.radius == c.radius);
}
```

The this Keyword

- The <u>this</u> keyword is the name of a reference that refers to an object itself. One common use of the <u>this</u> keyword is reference a class's *hidden data fields*.
- Another common use of the <u>this</u> keyword to enable a constructor to invoke another constructor of the same class.

Reference the Hidden Data Fields

```java
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;
  }
}
```

```
Suppose that f1 and f2 are two objects of F.
F f1 = new F(); F f2 = new F();

Invoking f1.setI(10) is to execute
   this.i = 10, where this refers f1

Invoking f2.setI(45) is to execute
   this.i = 45, where this refers f2
```

Calling Overloaded Constructor

```
public class Circle {
  private double radius;

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

  public Circle() {
    this(1.0);
  }

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

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

this is used to invoke another constructor

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

Book Exercises

TRY at least
9.1, 9.7, 9.10 & 9.11