CS1150
Principles of Computer Science
Objects and Classes

Yanyan Zhuang

Department of Computer Science

http://www.cs.uccs.edu/~yzhuang

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. E.g., a student, a desk, a circle, a button can all be viewed as objects.
- 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/attributes*) with their current values.
  - The ***behavior*** of an object is defined by a set of methods.

Java's predefined objects

- We've used predefined objects:
  - Import the object's class
  - Create an object (aka instance) from that class
  - Use methods defined for that object

```
import java.util.Scanner;
public class Assignment {
  public static void main(String[] args) {
    // Create a Scanner object to read input from the user
    Scanner input = new Scanner (System.in);
    int d = input.nextInt();
  }
}
```

Object state and behavior

- An object has two important pieces: **state** and **behavior**
- State
  - The properties that define an object: **things an object has**
  - A "dog" object may have properties such as color, size, gender
- Behavior
  - The methods that define an object: **things an object does**
  - A "dog" object may have behaviors such as sleep, bark, sit, etc.
- See StudentApp1.java

Object vs. Class

- An object represents an entity like a car, house, circle, dog, cat, student, etc.
- A class is used to construct an object
  - It defines an object's attributes and behaviors: the **blueprint**
  - Just as you can create more than one building from a blueprint, you can create more than one object from a class
- **A class is not an object - it's used to construct an object**

Creating Classes (Blueprints)

```
class ClassName{
…
}
```

Example:

```
class Circle{
  double radius;
  …
}
```

Declaring Objects (instances of a class)

To declare an object:

```
ClassName objectName;
```

Example:

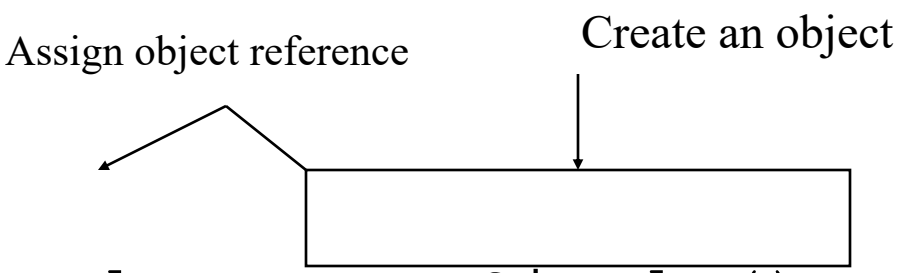
```
Circle myCircle;
Student myStudent;
```

To create an object:

```
new ClassName();
```

Declaring/Creating Objects in a Single Step

```
ClassName objectName = new ClassName();
```



Example:

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

See StudentApp2.java

How to organize class and test code

- Place the new class in the same file as the main method
  - Only one class contains the main method (test class)
  - The class with the main method needs to be used as the name of the .java file
  - Only one class can be declared as public
- Follow the examples…

Reference Data Fields

The data fields can be of primitive data types and reference types (arrays, Strings, and objects). The following Student class contains a data field “name” of String type.

```
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'
}
```

The default value of a data field is:

null for a String type, 0/0.0 for a numeric type, false for a boolean type, and '\u0000' for a char type.

Default Value for a Data Field

The default value of a data field is:

null for a String type, 0/0.0 for a numeric type, false for a boolean type, and '\u0000' for a char type.

```
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);
    System.out.println("isScienceMajor? " + student.isScienceMajor);
    System.out.println("gender? " + student.gender);
  }
}
```

Scope of Variables

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

- The scope of **instance** and **static** variables is the entire class. They can be declared anywhere inside a class.

Instance Variables and Static Variables

Instance variables belong to a **specific** instance / object: a circle's radius is 10, a student's age is 18

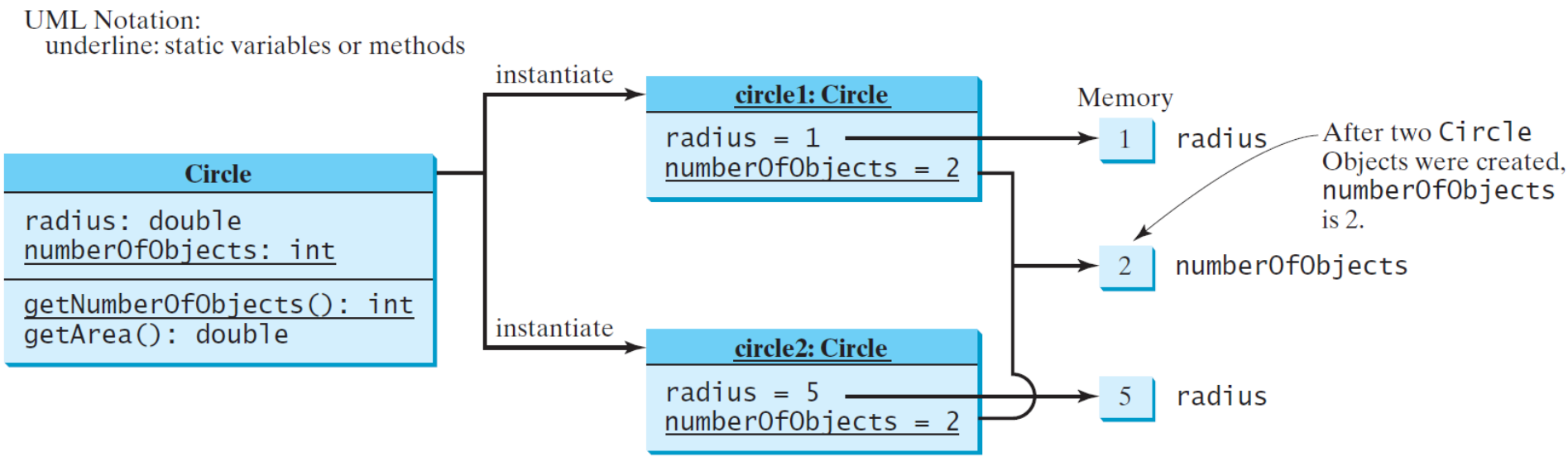
Static variables belong to an **entire** class

Static Variables, Constants (and Methods)

Static *variables* are **shared by all the instances of the class**. Static *constants* are final variables shared by all the instances of the class.

To declare static variables, constants, and methods, use the **static** modifier.

Static Variables



A static variable is stored at a **common location**

If **one** object changes its value, **all** objects gets the new value

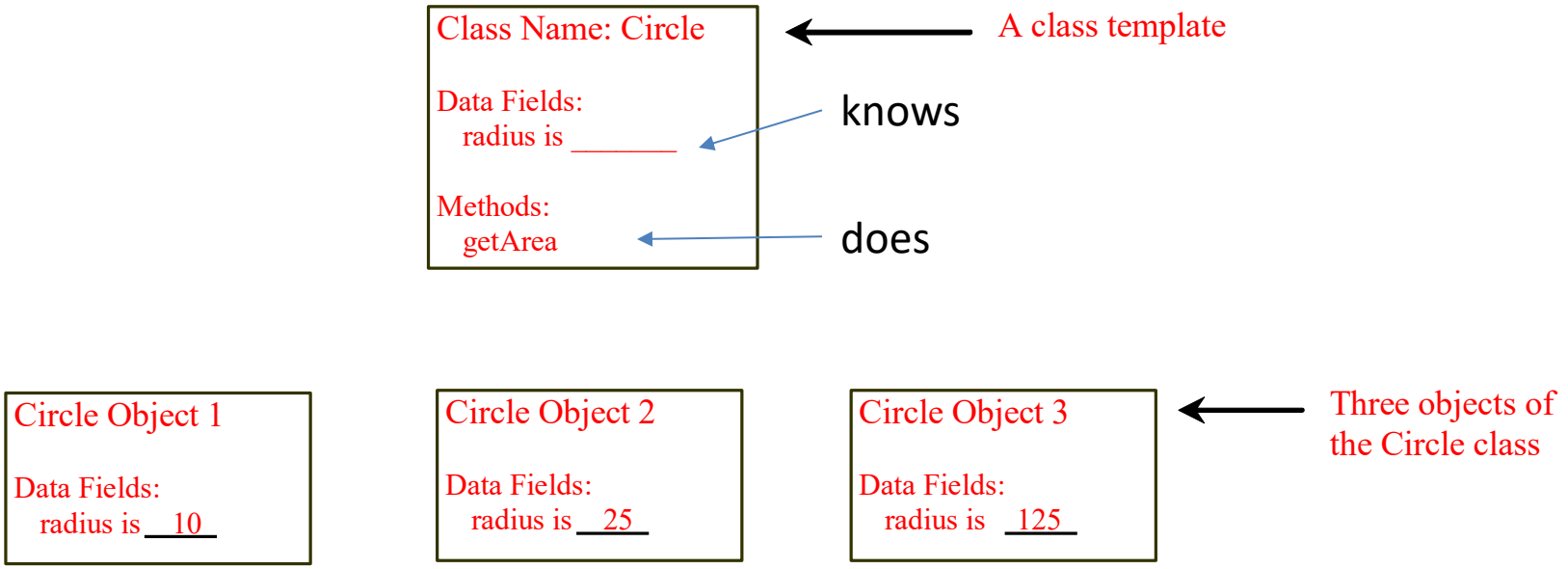
See StudentApp3.java (static var example)

Static Constants

- Constants of a class are shared by ALL objects of a class
- Constants by default use static keyword

```
// Declare constants a class
final static double PI = 3.14159;
```

Objects vs. Class



An object has both state and behavior

- The state defines the object: properties
- The behavior defines what the object does: methods
- Is a reference variable

See StudentApp4.java (method example)

Accessing Object's Members

- Referencing the object's properties (array's length):

  `objectName.data`

  e.g., `myCircle.radius`

- Invoking the object's method (String's length()):

  `objectName.methodName(arguments)`

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

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

Classes

Classes define objects of the same type (circles, students, etc).

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

A Java class provides a special type of methods, known as **constructors**, which are invoked to construct objects from the class.

Constructors

```
Circle() {
   …
}
new Circle();
```

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

Note: **no return value**!

```
Circle(double newRadius) {
   radius = newRadius;
   …
}
new Circle(5.0);
```

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.

A constructor with no parameters is referred to as a *no-arg constructor*.

Default 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, called ***a default constructor***, is provided automatically *only if no constructors are explicitly defined in the class*.

If we define any constructor at all, the default one is no longer available.

See StudentApp5.java, StudentApp6.java (constructor examples)

Constructors Summary

- A constructor has EXACTLY the same name as the class
- No return type is specified! Not even void
- ***NOT*** necessary to write a constructor for your classes!
  - Generally you will provide at least the no-arguments constructor - Cat()
  - If you don't - if you specify NO CONSTRUCTORS - Java automatically creates a ***default constructor***
    - ***It takes no arguments***
    - ***It has an empty body - no code!***
    - ***It does nothing to the instance variables***

Overloaded Constructors

- A constructor can be **overloaded** (StudentApp5)
  - public StudentD (){
    }
  - public StudentD (String lastName, String firstName){
    this.lastName = lastName;
    this.firstName = firstName;
    }
  - public StudentD (int ID, int level){
    this.studentID = ID;
    this.academicLevel = level;
    }

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 can be omitted

Practice

1. Create a class called Dog
2. Dog has attributes: name, breed, color (String) and age (int)
3. Create a dog object dog1, with no-arg constructor, and print its attributes
4. Assign dog1 with name Charlie, breed Husky, color white, age 2, then print its attributes

Practice

5. Create a static variable noOfDogs, increase it after creating dog1
6. Implement the print function in a method in the Dog class, called printMe(), and invoke it using the dog1 object
7. Create a no-arg constructor Dog(), and another Dog(String name, String breed)

Practice

8. Create another object dog2 with Dog("Max", "Lab"), increase noOfDogs after creating dog2, and invoke printMe()
9. Assign values to dog2's age (1), and color (gold), and invoke printMe()
10. Move Dog.noOfDogs++; in the constructors, print noOfDogs in main()

Array of Objects

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

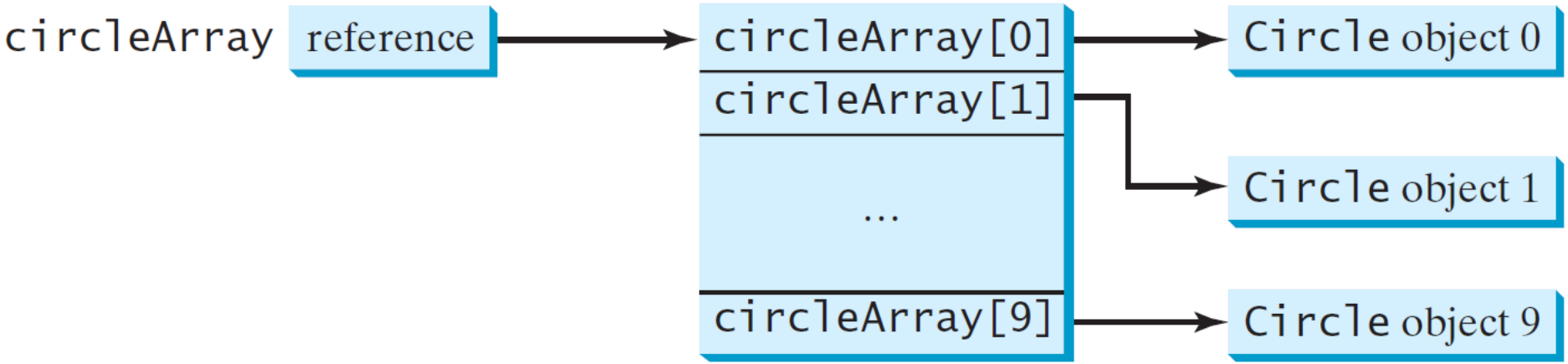
An array of objects is actually an *array of reference variables* (StudentApp6.java)

So invoking circleArray[1].toString() involves two levels of referencing:

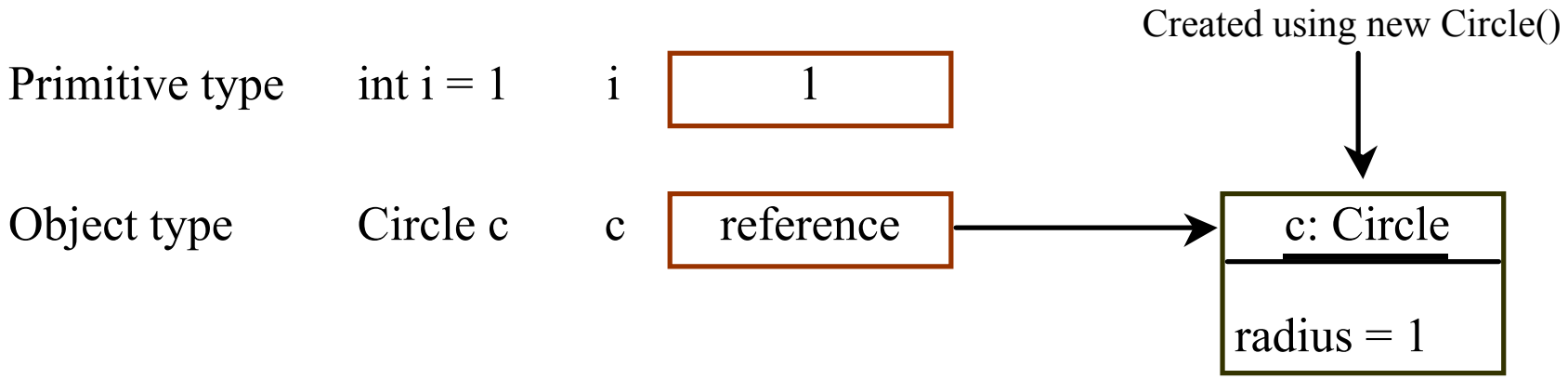
1. circleArray references to the entire array.
2. circleArray[1] references to a Circle object.

Array of Objects, cont.

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



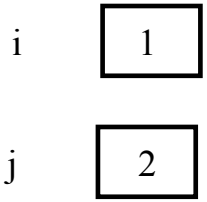
Differences between Variables of Primitive Data Types and Object Types



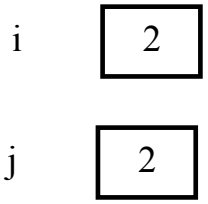
Copying Variables of Primitive Data Types and Object Types

Primitive type assignment int i = j

Before:

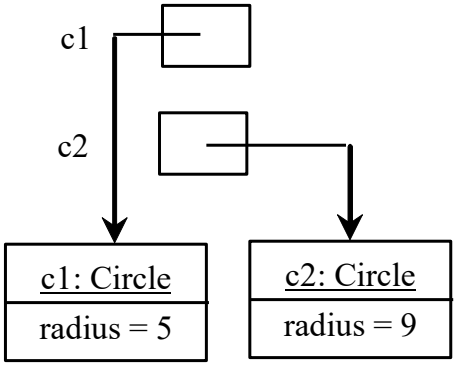


After:

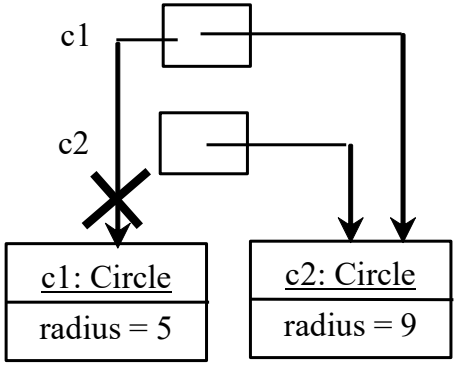


Object type assignment c1 = c2

Before:



After:



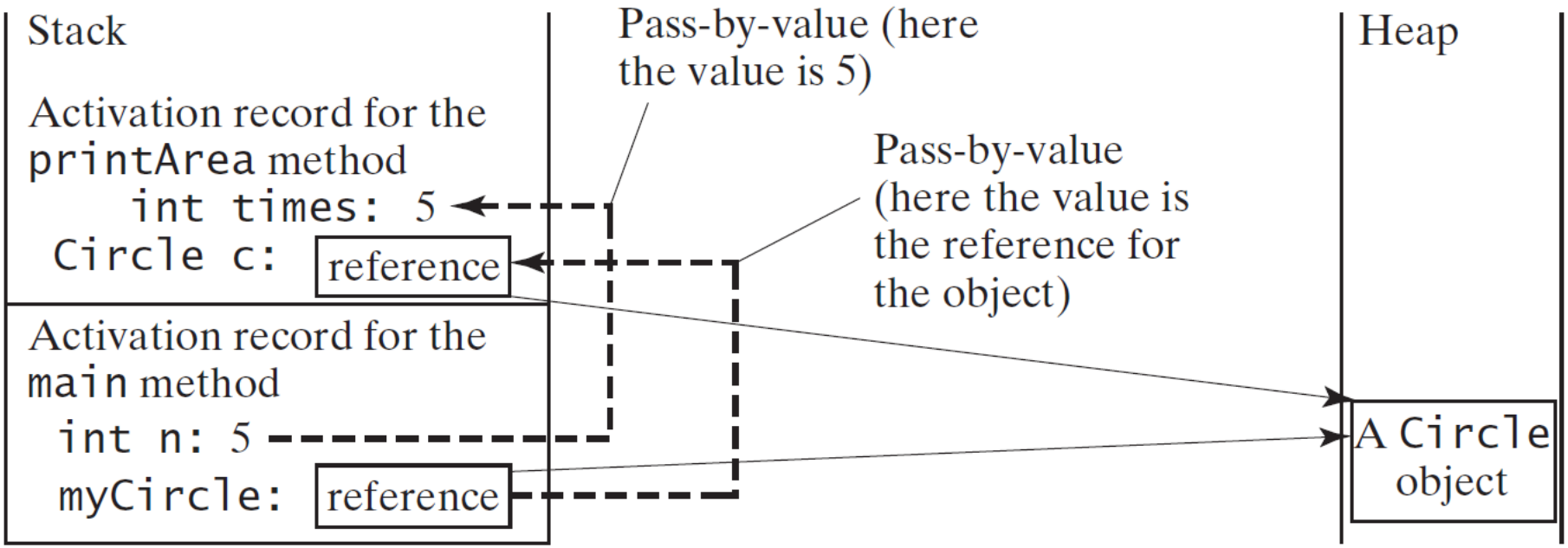
Objects and Reference Variables

- We first saw reference variables with arrays - an array *is* an object
  - Primitive Types
    - Declare an integer -> **int number = 5;**
    - The declaration indicates **type of data** you are declaring, and **allocates memory**
  - Array Types
    - Declare an array -> **int[] number;**
    - The declaration indicates
      - **type of data** the array will store, but **DOES NOT allocate memory**
      - Only creates a storage location for the reference (i.e. *number*) to the array
  - Object Types
    - Declare an object -> **Student student1;**
    - The declaration indicates
      - **type of data** you are declaring - a Student
      - **DOES NOT allocate memory**

Passing Objects to Methods

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

Passing Objects to Methods, cont.



Caution

Recall that we use

Math.methodName(arguments) (e.g., Math.pow(3, 2.5))

to invoke a method in the Math class. Can we invoke method() using Scanner.method()?

The answer is no. All Math methods are **static** methods, which are defined using the static keyword. However, method() without the static keyword is non-static. It must be invoked from an object using

objectRefVar.methodName(arguments) (e.g., myCircle.getArea()).

Static Methods

- A method that is shared by ALL objects of a class (also called a class method)
- Can be executed without the need to reference a particular instant of the class: e.g., Math class – Math.sqrt()
- Can use the object reference but, better to use class name because it makes it clear that the method is static
  - Can be done (will get a warning): int numOfStudents = student1. getNoOfStudents();
  - Better use class name to access the method: int numOfStudents = Student.getNoOfStudents();

See StudentApp7.java (static method example)

Note

- Use instance variables and instance methods when
  - When each object of the class needs an independent copy of a variable
  - When instance variables are accessed in methods the method must be an instance method
- Use static variables and static methods
  - When only one copy of the variable is needed and used by all objects
  - When all objects need to share a variable
  - When a method is not dependent on a specific instance

Practice

1. Continue with the Dog class example, implement a method setNoOfDogs() to increment the static variable noOfDogs by 1 (should this method be static or instance?)
2. Invoke setNoOfDogs() in the two constructors to automatically increment noOfDogs when an object is created
3. Implement a method getNoOfDogs() to get the total number of dog objects in the Dog class (should this be static or instance?)
4. How to invoke getNoOfDogs() in main() to query the total number of dogs?

Visibility Modifiers

By **default**, a class, variable, or method can be accessed by any class in the same package.

- `public`

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

- `private` See StudentApp7-8.java (private example)

  The data or methods can be accessed only by the declaring class.

The getter and setter methods are used to read and modify private properties.

```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 (i.e., no 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, as in (b).

It is OK if the object is declared in its own class, as 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**.

Why Data Fields Should Be private?

To protect data.

To make code easy to maintain.

Practice

1. Continue with the Dog class example, change the static variable noOfDogs to private: what else must be changed to make the code correct?
2. Should the method setNoOfDogs() be public or private? Why?
3. Should the method getNoOfDogs() be public or private? Why?
4. How to invoke getNoOfDogs() in main() to query the total number of dogs?

Summary

- Build your own classes and objects
- Constructors
- Static variables and methods