Re: Objectives in Arrays

- To describe why an array is necessary in programming
- To learn how to declare array reference variables and create arrays
- To initialize the values in an array
- To simplify programming using JDK 1.5 enhanced for loop
- To copy contents from one array to another
- To develop and invoke methods with array arguments and return type
- To sort an array using the selection sort algorithm
- To search elements using the linear or binary search algorithm
- To declare and create multidimensional arrays
Objectives in Objects & Classes

- To understand objects and classes and use classes to model objects
- To learn how to declare a class and how to create an object of a class
- To understand the roles of constructors and use constructors to create objects
- To distinguish between object reference variables and primitive data type variables
- To use classes in the Java library
- Data encapsulation: to declare private data fields with get and set methods
- To develop methods with object arguments
- To understand the difference between instance and static variables and methods
- To determine the scope of variables in the context of a class
- To use the keyword this as the reference to the current object that invokes the instance method
- To store and process objects in arrays
- To apply class abstraction to develop software

OO Programming Concepts

- Object-oriented programming (OOP): programming using objects.
- An object represents an entity in the real world that can be distinctly identified, a student, a desk, a circle, an animal, etc.
- An object has a unique identity, state, and behaviors.
- In OOP: state consists of a set of data fields (known as attributes/properties) with their current values. The behavior/activities of an object is defined by a set of methods.
- OOP: thinking in terms of objects
  - A Java program is a collection of cooperative objects
**Classes**

- *Classes* are constructs that define objects of the same type
  - Example: three circles with different radius values.

A Java class uses variables to define data fields, methods to define behaviors, and a special type of methods to construct objects from the class.

- fields of information: defines the state of the object
- commands: operations that change the internal state of the object; defined by methods with a void return type
- queries: operations that return state information about the object without changing its internal state; defined by methods with a specific non-void return type
- constructors: special methods that create and initialize new instances of a class; *have no return type and always bear the name of the class.*

---

**The Circle Class**

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;
    }
}
• UML: unified modeling language to describe the class template and the objects

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

Constructors

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

```
Circle() {
    // constructor body
}

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

• Constructors must have the same name as the class itself; can be overloaded.
• Constructors do not have a return type—not even void; otherwise?
• 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.
• What if you forgot to define a constructor?
Declaring Object Reference Variables

To reference an object, assign the object to a reference variable. Arrays are treated as objects in Java.

To declare a reference variable, use the syntax:

```java
ClassName objectRefVar;
```

Example:

```java
Circle myCircle;
```

Declaring/Creating Objects in a Single Step

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

Example:

```java
Circle myCircle = new Circle();
double myList = new double[10];
```

Arrays are treated as objects in Java.
Accessing Objects

- Both data and methods of an object are accessed via the reference variable

- Referencing the object's data:
  
  \[ \text{objectRefVar.data} \]
  
  e.g., \text{myCircle.radius}

- Invoking the object's method:
  
  \[ \text{objectRefVar.methodName(arguments)} \]
  
  e.g., \text{myCircle.getArea()}

A Simple Circle Class

```java
public class TestCircle1 { // the main class
    public static void main(String[] args) {
        Circle1 myCircle = new Circle1(5.0); // what is the value of myCircle.getArea()?
        Circle1 yourCircle = new Circle1(); // what is the value of yourCircle.getArea()?
        yourCircle.radius = 100; // what is the value of yourCircle.getArea()?
    }
}

class Circle1 {
    double radius;

    Circle1() { /** Construct a circle with radius 1 */
        radius = 1.0;
    }

    Circle1(double newRadius) { /** Construct a circle with a specified radius */
        radius = newRadius;
    }

    double getArea() { /** Return the area of this circle */
        return radius * radius * Math.PI;
    } // the program contains two classes: only one can be a public class, or in
    // or saved in different files
```
Trace Code

```java
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;
```

See Objects in NetBeans Debugger

Reference Data Fields

- The data fields of a class can be of reference types.
- For example, the Student class contains a data field name of the String type.

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

There is no constructor; what happens if creating a Student object?

What are the default values of the fields if an object is created?
The null Value and the Default Values

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

- The default value of a data field of an object is null for a reference type, 0 for a numeric type, false for a boolean type, and '\u0000' for a char type.

- Java assigns no default value to a local variable inside a method.

```java
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);
    }
}
```

Caution

- Java assigns no default value to a local variable inside a method.

```java
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);
    }
}
```

What kind of errors you expect?

Compilation error: variables not initialized
Variables of Primitive Data Types and Object Types

### Primitive type assignment: \( i = j \)

- **Before:**
  - \( i = 1 \)
  - \( j = 2 \)

- **After:**
  - \( i = 2 \)
  - \( j = 2 \)

### Object type assignment: \( c_1 = c_2 \)

- **Before:**
  - \( c_1: \text{Circle} \) \( \text{radius} = 5 \)
  - \( c_2: \text{Circle} \) \( \text{radius} = 9 \)

- **After:**
  - \( c_1: \text{Circle} \) \( \text{radius} = 9 \)
  - \( c_2: \text{Circle} \) \( \text{radius} = 9 \)

---

What is garbage collection?
Java Library: The Date Class

Java provides a system-independent encapsulation of date and time in the `java.util.Date` class (also `System.currentTimeMillis()`).

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

  Constructs a Date object for the current time. Constructs a Date object for a given time in milliseconds elapsed since 01/01, 1970, GMT. Returns a string representing the date and time. Returns the number of milliseconds since 01/01, 1970, GMT. Sets a new elapse time in the object.

For example, the following code

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


Java Library: The Random Class

- `Math.random()` obtains a random double value between [0.0,1.0)
  - a + Math.random() * b: obtains a double between [a, a+b)
  - More powerfully, use the Java library’s Random class

- `java.util.Random
  +Random()
  +Random(seed: long)
  +nextInt(): int
  +nextInt(n: int): int
  +nextLong(): long
  +nextDouble(): double
  +nextFloat(): float
  +nextBoolean(): boolean

  Constructs a Random object with the current time as its seed. Constructs a Random object with a specified seed. Returns a random int value. Returns a random int value between 0 and n (exclusive). Returns a random long value. Returns a random double value between 0.0 and 1.0 (exclusive). Returns a random float value between 0.0F and 1.0F (exclusive). 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.

```java
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("From 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

Instance Variables, and Methods

° Instance variables belong to a specific object (instance)
° Instance methods are invoked by an object of the class.
° Example:

```
Circle circle1 = new Circle();
Circle circle2 = new Circle(5);
```

circle1.radius is independent of circle2.radius, stored in different memory locations
° What if there is some data shared by all objects of a class? – if one object changes its value all other objects can see the change.

    static variable and static methods
### Static Variables, Constants, and Methods

- Static variables are shared by all the instances of the class.
- Static methods are not tied to a specific object, can be called without creating an instance of the class, e.g., Math.pow(3, 2.5)
- Static constants are final variables shared by all the instances of the class.

### Instance and Static Variables and Methods

This example adds a static variable `numberOfObjects` to track the number of `Circle` objects created.

```java
public class Circle2 {
    double radius; /* The radius of the circle; instance variable */
    static int numberOfObjects = 0; /* The number of the objects created; static variable */

    Circle2() { /* Construct a circle with radius 1; */
        radius = 1.0;
        numberOfObjects++;
    }

    Circle2(double newRadius) { /* Construct a circle with a specified radius */
        radius = newRadius;
        numberOfObjects++;
    }

    static int getNumberOfObjects() { /* Return numberOfObjects; static method */
        return numberOfObjects;
    }

    double getArea() { /* Return the area of this circle; instance method */
        return radius * radius * Math.PI;
    }
}
```
public class TestCircle2 {
/** Main method */
    public static void main(String[] args) {
        Circle2 c1 = new Circle2();
        System.out.println("Before creating c2"); // what is c1.radius, c1.numberOfObjects?
        Circle2 c2 = new Circle2(5);
        // Change the radius in c1
        c1.radius = 9;
        // Display c1 and c2 AFTER c2 was created
        System.out.println("After creating c2 and modifying " + "c1's radius to 9");
        System.out.println("c1 is : radius (" + c1.radius + ") and # of Circle objects (" + c1.numberOfObjects + ");
        System.out.println("c2 is : radius (" + c2.radius + ") and number of Circle objects (" + c2.numberOfObjects + ");
    }
}

See Objects in NetBeans Debugger

---

**Examples of Static Methods**

° Math.pow(3, 2.5);
   • All methods in the Math class are static methods.

° JOptionPane.showMessageDialog(null, "Hello, World!");

° Static methods (and variables): you do not have to create an object (instance) to use; though you can, but recommend using className.method and className.variable for clarity.

° Instance methods (and variables): you have to create an object to use the methods.

° How to decide whether a variable or method should be static or instance?
   • dependent on a specific instance of the class or not?
   • Every circle has its own radius, and thus the getArea method
   • Random method in the Math class is not dependent on an instance
Method Invocation: Static vs. Instance

- Can we invoke an instance method or variable from a static method? N
- Can we invoke a static method or variable from an instance method? Y

// what is wrong in the following code?
public class Foo {
    public static void main(String[] args) {
        method1();
    }
    public void method1() {
        method2();
    }
    public static void method2() {
        System.out.println("What is radius " + c.getRadius());
    }
    Circle c = new Circle();
}

Example of Using Static or Instance Methods

Review 7.11: Add the static keyword in the place of ? If appropriate

public class Test {
    private int count;

    public ? void main(String[] args) {
        ...
    }

    public ? int getCount() {
        return count;
    }

    public ? int factorial (int n) {
        int result = 1;
        for (int i = 1; i <= n; i++)
            result *= i;
        return result;
    }
}
Visibility Modifiers and Accessor/Mutator Methods

- By default, the class, variable, or method can be accessed by any class in the same package (package-private or package-access)
  - 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.

Examples

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

```java
public class Foo {
    private boolean x;
    public static void main(String[] args) {
        Foo foo = new Foo();
        System.out.println(foo.x);
    }
}

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

(a) This is OK because object foo is used inside the Foo class.

(b) This is wrong because x and convert are private in Foo.

---

Why Data Fields Should Be private?

Allowing data fields of a class be changed directly in the instances (of client programs) is dangerous, not a good programming practice!

```java
public class TestCircle2 {
    /** Main method */
    public static void main(String[] args) {
        Circle2 c1 = new Circle2();
        System.out.println("Before creating c2");
        Circle2 c2 = new Circle2(5);
        c1.radius = -9; // what if we want to ensure the radius is non-negative?
        c2.numberOfObjects = 5; // what if a programming bug here?
        System.out.println("After creating c2 and modifying " + "c1's radius to 9");
        System.out.println("c1 is : radius (" + c1.radius + ") and # of Circle objects (" + c1.numberOfObjects + ")");
        System.out.println("c2 is : radius (" + c2.radius + ") and number of Circle objects (" + c2.numberOfObjects + ")");
    }
}
```
Data Encapsulation and How It Works

- Data field encapsulation is to declare the field of a class private using the `private` modifier so as to prevent the direct modifications of the properties.
  - To protect data
  - To make class easy to maintain

- A private data field cannot be accessed by an object through a direct reference outside the class that defines the private field. How to allow a client to retrieve and modify a data field?
  - `get` (accessor): a query method
    
    Signature:
    ```java
    public returnType getPropertyName()
    ```

  - `set` (mutator): a command method
    
    Signature:
    ```java
    public void setPropertyName(dataType propertyValue)
    ```

Example of Data Field Encapsulation

```java
public class Circle3 {
    private double radius = 1; /** The radius of the circle */
    private static int numberOfObjects = 0; /** The number of the objects created: private & static */

    public Circle3() { /** Construct a circle with radius 1 */
        numberOfObjects++;
    }

    public Circle3(double newRadius) { /** Construct a circle with a specified radius */
        radius = newRadius;
        numberOfObjects++;
    }

    public double getRadius() { /** Query/accessor method: return radius */
        return radius;
    }

    public void setRadius(double newRadius) { /** Command/mutator method: set a new radius */
        radius = (newRadius >= 0) ? newRadius : 0;
    }

    public static int getNumberOfObjects() { /** Query/accessor method: return numberOfObjects */
        return numberOfObjects; /** how to make numberOfObjects value be 100? */
    }

    public double getArea() { /** Query/accessor method: return the area of this circle */
        return radius * radius * Math.PI;
    }
}
```
// TestCircle3.java: Demonstrate private modifier

class TestCircle3 {
    /** Main method */
    public static void main(String[] args) {
        // Create a Circle with radius 5.0
        Circle3 myCircle = new Circle3(5.0); // typo in the textbook
        System.out.println("The area of the circle of radius " +
            myCircle.getRadius() + " is " + myCircle.getArea());

        // Increase myCircle's radius by 10%
        myCircle.setRadius(myCircle.getRadius() * 1.1);
        System.out.println("The area of the circle of radius " +
            myCircle.getRadius() + " is " + myCircle.getArea());
    }
}

// what if myCircle.getRadius() is replaced by myCircle.radius?
// note that TestCircle3 and Circle3 are two different public classes

---

**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.

- How to make the previous Circle3 class immutable?

- Is a class with all private data fields and without mutators necessarily immutable?
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!
    }
}

Is Student class immutable or not?

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 (methods that return primitive data fields are fine).
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, object reference and arrays (the value is the reference to the object)

```java
public class TestPassObject {
    public static void main(String[] args) {
        Circle3 myCircle = new Circle3();
        printCircle(myCircle);
    }

    public static void printCircle(Circle3 c) {
        System.out.println("The area" + c.getRadius() + c.getArea());
    }
}
```

Re: Scope of Local Variables in Methods

° The scope of a variable: the part of the program where the variable can be referenced
  • a local variable: a variable defined inside a method
  • 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 declared before it can be used.

```java
public static void method1() {
    int k;
    ...
    for (int i = 1; i < 10; i++) {
        ...
        int j;  // What is the scope of i?
        ...
    }
}
```
**Scope of Variables in a Class**

- The scope of instance and static variables is the entire class. They can be declared anywhere (in any order) inside a class.
- Exception: when a data field is initialized based on a reference to another data field, the other field must be declared first.

```java
public class Circle {
    private double radius = 1;
    public double getArea() {
        return radius * radius * Math.PI;
    }
    public Circle(double radius) {
        this.radius = radius;
    }
}
```

```java
public class Foo {
    private int i;
    private int j = i + 1;
    // what is j's value?
    // if an object of Foo // is created
}
```

**The `this` Keyword**

- Hidden variables: if a parameter name of a class’s method is the same as a data field name of the class
  - Hidden instance variable and hidden static variable
- Use `ClassName.StaticVariable` to refer to a hidden static variable
- Use `this` to refer to an instance data field.
- Use `this` to the proxy of the object that invokes the instance method
- Use `this` to invoke an overloaded constructor of the same class.

```java
public class Circle {
    private double radius;
    public Circle(double radius) {
        this.radius = radius;
    }
    public Circle() {
        this(1.0);
    }
}
```
Serving as Proxy to the Calling Object

```java
class Employer {
    private int workAge = 1;
    private static double averageSalary = 10000;

    void setAge(int workAge) {
        this.workAge = workAge;
    }

    static void setSalary(double averageSalary) {
        Employer.averageSalary = averageSalary;
    }
}
```

Suppose that f1 and f2 are two objects of Employer.
Invoking f1.setAge(2) is to execute
f1.workAge = 2, where this is replaced by f1
Invoking f2.setAge(5) is to execute
f2.workAge = 5, where this is replaced by f2
Invoking f2.setSalary(50000) is to execute
Employer.averageSalary = 50000, to change the
t Value of the static variable

Array of Objects

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

- An array of objects is actually an array of reference variables.
- Invoking circleArray[1].getArea() involves two levels of referencing.
  - circleArray references to the entire array
  - circleArray[1] references to a Circle object.
Programming Example: Array of Objects

° Create an array of Circle3 circles with different radius \([0, 100)\), print the area of each circle, and summarize the areas of all circles

° In a Java program, we can put multiple classes in one file (or packaging), but only one class can be a public class. The public class must have the same name as the file name, and the main method must be in a public class.

```java
public class TotalArea {
    public static void main(String[] args) {
        Circle3[] circleArray;
        int number = 10;
        circleArray = createCircleArray(number);
        printCircleArray(circleArray);
    }
    public static Circle3[] createCircleArray(int number) {
        ....
    }
    public static void printCircleArea(Circle3[] circleArray) {
        ....
    }
    public static double sum(Circle3[] circleArray) {
        ....
    }
}
```

Class Abstraction and Encapsulation

° Recall method abstraction

° Class abstraction means to separate class implementation from the use of the class.

° The creator of the class provides a description of the class and let the user know how the class can be used.

° The user of the class does not need to know how the class is implemented. The detail of implementation is encapsulated and hidden from the user.

Class implementation is like a black box hidden from the clients

Class Contract (Signatures of public methods and public constants)

UML

Unified modeling language

Clients use the class through the contract of the class
Example: The Loan Class

<table>
<thead>
<tr>
<th>Loan</th>
</tr>
</thead>
<tbody>
<tr>
<td>The annual interest rate of the loan (default: 2.5).</td>
</tr>
<tr>
<td>The number of years for the loan (default: 1).</td>
</tr>
<tr>
<td>The loan amount (default: 1000).</td>
</tr>
<tr>
<td>The date this loan was created.</td>
</tr>
</tbody>
</table>

+Loan()                                    
Constructs a default Loan object.

+Loan(annualInterestRate: double, 
     numberOfYears: int, 
     loanAmount: double)  
Constructs a loan with specified interest rate, years, and loan amount.

+getAnnualInterestRate(): double          
Returns the annual interest rate of this loan.

+getNumberOfYears(): int                  
Returns the number of the years of this loan.

+getLoanAmount(): double                  
Returns the amount of this loan.

+getLoanDate(): Date                     
Returns the date of the creation of this loan.

+setAnnualInterestRate( 
    annualInterestRate: double): void    
Sets a new annual interest rate to this loan.

+setNumberOfYears( 
    numberOfYears: int): void            
Sets a new number of years to this loan.

+setLoanAmount( 
    loanAmount: double): void             
Sets a new amount to this loan.

+getMonthlyPayment(): double              
Returns the monthly payment of this loan.

+getTotalPayment(): double                
Returns the total payment of this loan.

Example: The TestLoan Class

Based on the UML of the Loan class in the previous slide, write a test program, TestLoan, that uses the Loan class without knowing how the Loan class is implemented: reads in the interest rate, payment period (in years), and load amount; creates a Loan object; and then obtains the monthly payment and total payment using the instance methods in the Loan class.

Demonstrating that developing a class and using a class are two separate tasks

developing an application without knowing the complex implementation of certain classes
Example: The Course Class

<table>
<thead>
<tr>
<th>Course</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-name: String</td>
<td>The name of the course.</td>
</tr>
<tr>
<td>-students: String[]</td>
<td>The students who take the course.</td>
</tr>
<tr>
<td>-numberOfStudents: int</td>
<td>The number of students (default: 0).</td>
</tr>
<tr>
<td>+Course(name: String)</td>
<td>Creates a Course with the specified name.</td>
</tr>
<tr>
<td>+getName(): String</td>
<td>Returns the course name.</td>
</tr>
<tr>
<td>+addStudent(student: String): void</td>
<td>Adds a new student to the course list.</td>
</tr>
<tr>
<td>+getStudents(): String[]</td>
<td>Returns the students for the course.</td>
</tr>
<tr>
<td>+getNumberOfStudents(): int</td>
<td>Returns the number of students for the course.</td>
</tr>
</tbody>
</table>

Reading

- Chapter 7 of the textbook: 7.1 – 7.16
- Preview Chapter 8: String and Text I/O
- Do review questions: 7.1-7.3, 7.5, 7.9-7.23