Lecture 23:
Methods and Arrays
Begin Classes and Objects

Mar 24 2015

Slides adapted from lectures by Dr. Anca Doloc-Mihu
A note on sorting

See Resources from Thursday March 19th
For descriptions / discussions of
- Selection sort
- Insertion sort
- Bubble sort

You’ll need to be familiar with these for the quiz!!!
Passing Arrays to Methods

```java
public static void printArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
}
```

Invoke the method - Reference passing
```java```
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```java```

Invoke the method – No Reference passing
```java```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```java```

Anonymous array
Anonymous Array

The statement

```java
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

creates an array using the following syntax:

```java
new dataType[]{literal0, literal1, ..., literalk};
```

There is no explicit reference variable for the array. Such array is called an *anonymous array*. 
Pass By Value

Java uses *pass by value* to pass arguments to a method. There are important differences between passing a value of variables of primitive data types and passing arrays (or any non-primitive datatype).

- For a parameter of a primitive type value, *the actual value is passed*. Changing the value of the local parameter inside the method does not affect the value of the variable outside the method.

- For a parameter of an array type, the value of the parameter contains a reference to an array; *this reference is passed to the method*. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.
Returning an Array from a Method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

//invoke method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
Trace the reverse Method

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

Declare result and create array

**list**

| 1 | 2 | 3 | 4 | 5 | 6 |

**result**

| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

i = 0 and j = 5
Assign list[0] to result[5]
Trace the reverse Method, cont.

```
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 1 and j becomes 4
Trace the reverse Method, cont.

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```

list | 1 2 3 4 5 6
result | 0 0 0 0 0 1
```

```
Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
  1  2  3  4  5  6
   ^     
```

```
list
```

```
  0  0  0  0  2  1
     ^
```

```
result
```

```
i = 1 and j = 4
Assign list[1] to result[4]
```
Trace the reverse Method, cont.

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}

After this, i becomes 2 and j becomes 3

<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

14
Trace the reverse Method, cont.

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

List:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>

Result:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>

The loop starts at index 0 and iterates to the end of the list. At each iteration, the value at the current index is copied to the corresponding position in the result array, moving from the end to the beginning. The loop condition `i < list.length` ensures that the loop continues as long as `i` is less than the length of the list.

For example, when `i` is 2, it is still less than 6, and the value at index 2 in `list` is copied to index 3 in `result`.

Output:

```
list: 6 5 4 3 2 1
result: 1 2 3 4 5 6
```
Trace the reverse Method, cont.

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}

i = 2 and j = 3
Assign list[i] to result[j]

<table>
<thead>
<tr>
<th>list</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6</td>
<td>0 0 0 3 2 1</td>
</tr>
</tbody>
</table>

16
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    
    return result;
}

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

After this, i becomes 3 and j becomes 2

<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Trace the reverse Method, cont.

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--)
        result[j] = list[i];
    return result;
}
```

<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>result</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>

i (=3) is still less than 6
Trace the reverse Method, cont.

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
Trace the reverse Method, cont.

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```java
class Solution {
    public static int[] reverse(int[] list) {
        int[] result = new int[list.length];
        for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
            result[j] = list[i];
        }
        return result;
    }
}
```

### Example

```
list
1 2 3 4 5 6

result
0 0 4 3 2 1
```

After this, i becomes 4 and j becomes 1.
Trace the reverse Method, cont.

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Trace the reverse Method, cont.

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```
Trace the reverse Method, cont.

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

After this, i becomes 5 and j becomes 0

### list

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

### result

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

23
Trace the reverse Method, cont.

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}

i (=5) is still less than 6

<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Trace the reverse Method, cont.

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

---

**Trace:**

1. Initialize `result` array with same length as `list`.
2. Initialize `i` to 0 and `j` to `result.length - 1`.
3. Loop through `list` from `i = 0` to `list.length - 1`.
   - Assign `list[i]` to `result[j]`.
4. Return `result`.

**Example:**

- `list1 = {1, 2, 3, 4, 5, 6}`
- `list2 = reverse(list1)`

<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- `i = 5` and `j = 0` Assign `list[i]` to `result[j]`
Object-Oriented Programming
Contents

• Defining classes
• Creating objects
• Constructors
• Declaring objects
• Accessing object’s members
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 graphical user interfaces and large scale software systems. Suppose you want to develop a graphical user interface as shown below. How do you program it?
Object-oriented programming (OOP) involves programming using objects.

An object represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan 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) with their current values.

The behavior of an object is defined by a set of methods.
Objects

An object has both a state and behavior. The state defines the object, and the behavior defines what the object does.
Classes are constructs 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, which are invoked to construct objects from the class.

An object is an instance of a class. Creating instances is referred to as instantiation.
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;
   }
}

32
public class TestCircle {

    public static void main() {
        Circle circle1 = new Circle(); // default radius = 1
        System.out.println("Radius = " + circle1.radius + " area = " + circle1.getArea());

        Circle circle2 = new Circle(25); // radius = 25
        System.out.println("Radius = " + circle2.radius + " area = " + circle2.getArea());

        Circle circle3 = new Circle(125); // radius = 125
        System.out.println("Radius = " + circle3.radius + " area = " + circle3.getArea());
    }
}

Radius = 1.0 area = 3.14159
Radius = 25.0 area = 1963.49540
Radius = 125.0 area = 49087.38521
UML Class Diagram

Circle
radius: double
Circle()
Circle(newRadius: double)
getArea(): double

Class name
Data fields
Constructors and methods

circle1: Circle
radius = 1.0

circle2: Circle
radius = 25

circle3: Circle
radius = 125

UML notation for objects

UML – Unified Modeling Language
Example: Defining Classes and Creating Objects
Objective: Demonstrate creating objects, accessing data, and using methods.

class Person{//file Person.java
    String name = "something";

    Student(String newname){
        name = newname;
    }

    String getPersonName(){
        return name;
    }
}

//in main() in some other file
Person st = new Person("Ana"); // created object Person with name "Ana"
System.out.println("Person in room: ", st.getPersonName());
Constructors

Constructors are a special kind of methods that are invoked to construct objects by using the new operator.

```java
Circle() {
}
```

```java
Circle(double newRadius) {
    radius = newRadius;
}
```
Constructors, cont.

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

• 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.
Creating Objects Using Constructors

new ClassName();

Example:
new Circle();

new Circle(5.0);
Default Constructor

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

```java
Circle() {
}
```

(Default) Constructor

- A default constructor is the one that has no arguments and is provided automatically for all classes. This constructor will initialize all instance variables to default values.
- However, if the developer provides a constructor, the compiler's default constructor is no longer added.
- The developer will need to explicitly add a default constructor.

- It is a good practice to always have a default, no-argument constructor.
Declaring Object Reference Variables

To reference an object, assign the object to a reference variable.

To declare a reference variable, use the syntax:

```
ClassName objectRefVar;
```

Example:

```
Circle myCircle;
```
Declaring/Creating Objects in a Single Step

ClassName objectRefVar = new ClassName();

Example:
Circle myCircle = new Circle();

Newly created objects are allocated in the memory and can be accessed via reference variable (object’s reference variable).
Accessing Object’s Members

• Referencing the object’s data:
  \[ \text{objectRefVar.data} \]
  *e.g., myCircle.radius*

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

**A class is a reference type.** This means that a variable of the class type can reference an instance of the class.

Circle myCircle; //myCircle references a Circle object
Trace Code

```java
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;
```
Trace Code, cont.

```
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;
```
Trace Code, cont.

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

Assign object reference to myCircle

myCircle: Circle
radius: 5.0
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
Trace Code, cont.

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

Create a new Circle object

<table>
<thead>
<tr>
<th>: Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius: 5.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius: 1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>no value</td>
</tr>
</tbody>
</table>
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

Assign object reference to yourCircle
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

Change radius in yourCircle
Classes and Objects

- A Java program consists of one or more classes.
- A class is an abstract description of objects.
- Here is an example class:
  ```java
  class Dog {
    // description of a dog goes here...
  }
  ```
- Here are some objects of that class:

Fig. 1: Instantiating two Trees from the Tree class