CS062
DATA STRUCTURES AND ADVANCED PROGRAMMING

2: Java Basics

Alexandra Papoutsaki
she/her/hers

Tom Yeh
he/him/his
Lecture 2: Java Basics

- Methods
- Arrays
- Operators
- Control Flow

Some slides adopted from Algorithms, 4th Edition and Oracle tutorials
Methods

- A collection of grouped statements that perform a logical operation and control the behavior of objects.
- By convention method names should be a verb (+ noun) in lowercase.
- Syntax: modifier returnType methodName(type parameter-name,…){…}
  - E.g., `public int getCadence(){…return cadence;}`
- Signature: method name and the number, type, and order of its parameters.
- Control goes back to the calling program as soon as a `return` statement is reached. If it does not return anything it is `void`.
- Can also be `static`, therefore shared by all instances of a class.
- Can be overloaded (same name, different parameters).
**Constructors** are invoked to create objects from class blueprints

- Constructor declarations look like method declarations but have the same name with the class and no return type

```java
// the Bicycle class has one constructor
public Bicycle(int startCadence, int startSpeed, int startGear) {
    gear = startGear;
    cadence = startCadence;
    speed = startSpeed;
}
```

- To instantiate a new object use the `new` keyword

```java
Bicycle myBike = new Bicycle(30, 0, 8);
```

- A class can have multiple constructors, including a no-argument constructor

```java
// the Bicycle class could have a no-argument constructor
public Bicycle() {
    gear = 1;
    cadence = 10;
    speed = 0;
}
```

```java
Bicycle yourBike = new Bicycle();
```

**YOU DON’T HAVE TO PROVIDE A CONSTRUCTOR BUT IT’S ALWAYS A GOOD IDEA TO DO SO**
this keyword

- Within an instance method or a constructor used to refer to current object.
- Can be used to call instance variables, methods and constructors. E.g.,

```java
public class Point {
    private int x = 0;
    private int y = 0;

    //constructor
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```
this keyword to invoke constructors

```java
public class Rectangle {
    private int x, y;
    private int width, height;

    public Rectangle() {
        this(0, 0, 1, 1);
    }

    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }

    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
}
```
Parameters

- Variables passed in a method definition. You need to specify their type. E.g.,

  ```java
  int countToNumber(int number) {
      //...
  }
  ```

- The arguments are the data you pass into the method's parameters. E.g., `countToNumber(3);`
Combination of instance/static variables/methods

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access static variables and static methods directly.
- Static methods can access static variables and static methods directly.
- Static methods **cannot** access instance variables or instance methods directly—they must use an object reference.
  - E.g., “Cannot make a static reference to the non-static field” in main method
- Static methods cannot use the `this` keyword as there is no instance of an object for `this` to refer to.
Lecture 2: Java Basics

- Methods
- Arrays
- Operators
- Control Flow
Array: Our first data structure

- Container object that holds a sequence of a fixed number of values of the same type.
- The length of the array is established during its creation and stays fixed.
- Each item is called an element and each element is accessed by its index.
- If we have $N$ elements the indices range from $0...N - 1$. 


Creating and initializing an array

1. Declare the array name and the type of its elements. E.g., `double[] a;`

2. Create the array. E.g., `a = new double[N];`

3. Initialize the array values. E.g.,
   ```java
   for (int i = 0; i < N; i++){
       a[i] = 10.0;
   }
   ```

   - Default array initialization: We can combine all three steps into a single statement and all elements will take the default values (0, `false`, or `null` depending on type). E.g., `double[] a = new double[N];`

   - Initializing declaration: List literal values between curly braces, separated by comma. E.g., `int[] b = {1,2,3};`
Using arrays

- Arrays have fixed size. We can access this size through its instance variable `length` (tsk, tsk, Java). E.g., `a.length`

- You can access or change an element using the `a[i]` notation.

- If you request an index that is either negative or larger than `length-1`, then you will get an `ArrayIndexOutOfBoundsException`. 
Multidimensional arrays

/**
 * Illustration of multidimensional arrays
 *
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html
 *
 */
public class MultiDimArrayDemo {
    public static void main(String[] args) {
        String[][][] names = {
            {"Mr. ", "Mrs. ", "Ms. "},
            {"King", "Park"}
        };
        // Mr. King
        System.out.println(names[0][0] + names[1][0]);
        // Mrs. Park
        System.out.println(names[0][1] + names[1][1]);
        // Ms. King
        System.out.println(names[0][2] + names[1][0]);
    }
}
Aliasing

- An array name refers to the whole array – if we assign one array name to another, then both refer to the same array.

- This can lead to aliasing problems.

```java
int[] a = new int[N];
a[i] = 1234;
int[] b = a;
b[i] = 5678;  // a[i] is now 5678.
```
Practice Time:

1. The term "instance variable" is another name for ___.
2. The term "class variable" is another name for ___.
3. A local variable stores temporary state; it is declared inside a ___.
4. A variable declared within the opening and closing parentheses of a method signature is called a _____. The actual value passed is called an ___.
5. What are the eight primitive data types supported by the Java programming language?
6. Character strings are represented by the class ___.
7. An ___ is a container object that holds a fixed number of values of a single type.
Answers:

1. The term "instance variable" is another name for **non-static/member field**.
2. The term "class variable" is another name for **static field**.
3. A local variable stores temporary state; it is declared inside a **method**.
4. A variable declared within the opening and closing parentheses of a method is called a **parameter**. The actual value passed is called an argument.
5. What are the eight primitive data types supported by the Java programming language? **byte, short, int, long, float, double, boolean, char**
6. Character strings are represented by the class **java.lang.String**.
7. An **array** is a container object that holds a fixed number of values of a single type.

[https://docs.oracle.com/javase/tutorial/java/nutsandbolts/QandE/answers_variables.html](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/QandE/answers_variables.html)
Lecture 2: Java Basics

- Methods
- Arrays
- Operators
- Control Flow
## Operator precedence

<table>
<thead>
<tr>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>postfix</td>
<td>expr++ expr--</td>
</tr>
<tr>
<td>unary</td>
<td>/+-+expr -/---expr !boolean</td>
</tr>
<tr>
<td>multiplicative</td>
<td>* / %</td>
</tr>
<tr>
<td>additive</td>
<td>+ -</td>
</tr>
<tr>
<td>relational</td>
<td>&lt; &gt; &lt;= &gt;= instanceOf</td>
</tr>
<tr>
<td>equality</td>
<td>== !=</td>
</tr>
<tr>
<td>logical AND</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>logical OR</td>
<td></td>
</tr>
<tr>
<td>assignment</td>
<td>= += -= *= /=</td>
</tr>
</tbody>
</table>
Assignment operator

- = assigns the value on its right to the operand on its left
  - e.g., `int cadence = 3;`
Arithmetic operators

/**
 * Illustration of the arithmetic operators
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
 */

public class ArithmeticDemo {

    public static void main(String[] args) {

        int result = 1 + 2;
        // result is now 3
        System.out.println("1 + 2 = " + result);
        int original_result = result;

        result = result - 1;
        // result is now 2
        System.out.println(original_result + " - 1 = " + result);
        original_result = result;

        result = result * 2;
        // result is now 4
        System.out.println(original_result + " * 2 = " + result);
        original_result = result;

        result = result / 2;
        // result is now 2
        System.out.println(original_result + " / 2 = " + result);
        original_result = result;

        result = result + 8;
        // result is now 10
        System.out.println(original_result + " + 8 = " + result);
        original_result = result;

        result = result % 7;
        // result is now 3
        System.out.println(original_result + " % 7 = " + result);
    }
}

Output:

1 + 2 = 3
3 - 1 = 2
2 * 2 = 4
4 / 2 = 2
2 + 8 = 10
10 % 7 = 3
Unary operators require only one operand

```java
/**
 * Illustration of the unary operators
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
 */

public class UnaryDemo {

    public static void main(String[] args) {

        int result = +1;
        // result is now 1
        System.out.println(result);

        result--;
        // result is now 0
        System.out.println(result);

        result++;
        // result is now 1
        System.out.println(result);

        result = -result;
        // result is now -1
        System.out.println(result);

        boolean success = false;
        // false
        System.out.println(success);

        System.out.println(!success);
    }
}
```
The ++/-- operators can be applied pre or post operand

/**
 * Illustration of the prefix/postfix unary operator
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
 */

public class PrePostDemo {
    public static void main(String[] args) {
        int i = 3;
        i++;
        // prints 4
        System.out.println(i);
        ++i;
        // prints 5
        System.out.println(i);
        // prints 6
        System.out.println(++i);
        // prints 6
        System.out.println(i++);
        // prints 7
        System.out.println(i);
    }
}
Equality/Relational operators

/**
 * Illustration of the equality/relational operators
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
 */

public class ComparisonDemo {

    public static void main(String[] args) {
        int value1 = 1;
        int value2 = 2;
        if (value1 == value2) {
            System.out.println("value1 == value2");
        }
        if (value1 != value2) {
            System.out.println("value1 != value2");
        }
        if (value1 > value2) {
            System.out.println("value1 > value2");
        }
        if (value1 < value2) {
            System.out.println("value1 < value2");
        }
        if (value1 <= value2) {
            System.out.println("value1 <= value2");
        }
    }

}
Conditional operators

/**
 * Illustration of the equality/relational operators
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
 */

public class ConditionalDemo {

    public static void main(String[] args) {
        int value1 = 1;
        int value2 = 2;
        if ((value1 == 1) && (value2 == 2))
            System.out.println("value1 is 1 AND value2 is 2");
        if ((value1 == 1) || (value2 == 1))
            System.out.println("value1 is 1 OR value2 is 1");
    }
}

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A AND B</th>
<th>A OR B</th>
<th>NOT A</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
1. Consider the following code:
   ```java
   arrayOfInts[j] > arrayOfInts[j+1]
   ```
   Which operators does the code contain?

2. Consider the following code snippet:
   ```java
   int i = 10;
   int n = i++%5;
   ```
   a. What are the values of \(i\) and \(n\) after the code is executed?
   b. What are the final values of \(i\) and \(n\) if instead of using the postfix increment operator (\(i++\)), you use the prefix version (\(++i\))?

3. To invert the value of a boolean, which operator would you use?

4. Which operator is used to compare two values, = or == ?
Answers:

1. >, +

2.
   a. i is 11, and n is 0
   b. i is 11, and n is 1.

3. The logical complement operator!

4. ==

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/QandE/answers_operators.html
Lecture 2: Java Basics

- Methods
- Arrays
- Operators
- Control Flow
If-then statement

```java
public void applyBrakes() {
    // the "if" clause: bicycle must be moving
    if (isMoving){
        // the "then" clause: decrease current speed
        currentSpeed--;
    }
}
```
If-then-else statement

```java
/**
 * Illustration of the if then else control flow
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html
 */

public class IfElseDemo {
    public static void main(String[] args) {
        int testscore = 76;
        char grade;

        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

Once a condition is satisfied, the appropriate statements are executed and the remaining conditions are not evaluated.
While statement

/**
 * Illustration of the if then else control flow
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html
 *
 */
public class WhileDemo {

    public static void main(String[] args){
        int count = 1;
        while (count < 11) {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
For statement

```java
for (initialization; termination; increment) {
    statement(s)
}
```

/**
 * Illustration of the for loop
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html
 */

```java
public class ForDemo {
    public static void main(String[] args) {
        for (int i = 1; i < 11; i++) {
            System.out.println("Count is: " + i);
        }
    }
}
```

Enhanced for statement in most data structures

/**
 * Illustration of the enhanced for flow
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html
 *
 */
class EnhancedFor {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        for (int item : numbers) {
            System.out.println("Count is: " + item);
        }
    }
}
破断ステートメント

- 破断ステートメントをforまたはwhileループで終了するためのuse breakを設定することがあります。

```java
/**
 * Illustration of the break branch
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html
 */
public class BreakDemo {
    public static void main(String[] args) {
        int[] arrayOfInts = { 32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127 };
        int searchfor = 12;

        int i;
        boolean foundIt = false;

        for (i = 0; i < arrayOfInts.length; i++) {
            if (arrayOfInts[i] == searchfor) {
                foundIt = true;
                break;
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor + " at index " + i);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```
CONTROL FLOW

Continue statement

- Use **continue** to skip the current iteration of *for* or *while* loop.

* Illustration of the continue branch
*
* @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html
*
*/

class ContinueDemo {
    public static void main(String[] args) {
        String searchMe = "peter piper picked a " + "peck of pickled peppers";
        int max = searchMe.length();
        int numPs = 0;
        for (int i = 0; i < max; i++) {
            // interested only in p's
            if (searchMe.charAt(i) != 'p')
                continue;

            // process p's
            numPs++;
        }
        System.out.println("Found "+ numPs + " p's in the string.");
    }
}
Return statement

- The `return` statement exits from the current method, and control flow returns to where the method was invoked.
- Can return a value, e.g., `return counter++;`
- Or not, e.g., `return;`
Lecture 2: Java Basics

- Methods
- Arrays
- Operators
- Control Flow
ASSIGNED READINGS AND PRACTICE PROBLEMS

Readings:

- Oracle’s guides:
  - Language Basics: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html

- Textbook:
  - Chapter 1.1 (Pages 8–35)
  - Chapter 1.2 (Pages 64–77, 84–88, 96–99)

Practice Problems:

- 1.1.1–1.1.5, 1.1.8–1.1.12, 1.2.4, 1.2.8