Learning the Java Language

Based on The Java™ Tutorial
(http://docs.oracle.com/javase/tutorial/java/)

Bryn Mawr College
CS206 Intro to Data Structures

Language Basics

- Variables
- Operators
- Expressions, Statements and Blocks
- Control Flow Statements
What is an object

- Objects
  - State: stored in **fields**
  - Behavior: exposed through **methods**, the primary mechanism for object-to-object communication
- **Data encapsulation**: hiding internal state and requiring all interaction to be performed through an object’s methods (an OOP fundamental principle)
- Benefits of using objects
  - Modularity
  - Information-hiding
  - Code re-use
  - Pluggability and debugging ease

What is a class

- **Class**: A blueprint for a software object.
- A specific object is called an **instance** of the class of objects.

```java
public class Bicycle {
    int speed = 0;
    int gear = 1;
    
    void changeGear(int newValue) {
        gear = newValue;
    }
    
    void speedUp(int increment) {
        speed = speed + increment;
    }
    
    void applyBrakes(int decrement) {
        speed = speed - decrement;
    }
    
    void printStates() {
        System.out.println("speed:" + speed + " gear:" + gear);
    }
}

public class BicycleDemo {
    public static void main(String[] args) {
        // Create two different Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();
        
        // Invoke methods on those objects
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike1.printStates();
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.speedUp(10);
        bike2.changeGear(3);
        bike2.printStates();
    }
}
```
Variables

• An object stores its state in fields.
  o int speed = 0;
  o int gear = 1;

• Kinds of variables:
  o **Instance Variables (Non-Static Fields):** a.k.a instance variables (because their values are unique to each instance of a class, i.e., to each object. E.g., the currentSpeed of one bicycle is independent from the currentSpeed of another.
  o **Class Variables (Static Fields):**
    • Any field declared with the `static` modifier.
    • **Exactly** one copy of this variable in existence, regardless of how many times the class has been instantiated.
    • E.g., `static int numGears = 6;`
  o **Local Variables:** only visible to the methods in which they are declared; they are not accessible from the rest of the class.
  o **Parameters**
    • E.g., `void changeGear(int newValue)`

Variable Naming

• Variable names are case-sensitive.
• A variable name begins with:
  o Legally, a letter, the dollar sign "$", or the underscore character "_".
  o Convention: begins with a letter, not "$" or "_".
• Subsequent characters:
  o letters, digits, dollar signs, or underscore characters
• White space is not permitted.
• Must not choose keyword or reserved word.
• One word only: all lowercase letters.
• More than one word: capitalize the first letter of each subsequent word. E.g., `currentGear`
• If a variable stores a constant value, capitalize every letter. E.g., `static final int NUM_GEARs = 6.`
Primitive Data Types

Display 1.2 Primitive Types

<table>
<thead>
<tr>
<th>TYPE NAME</th>
<th>KIND OF VALUE</th>
<th>MEMORY USED</th>
<th>SIZE RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>true or false</td>
<td>1 byte</td>
<td>not applicable</td>
</tr>
<tr>
<td>char</td>
<td>single character</td>
<td>1 byte</td>
<td>all Unicode characters</td>
</tr>
<tr>
<td></td>
<td>(Unicode)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>integer</td>
<td>2 bytes</td>
<td>−128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>integer</td>
<td>2 bytes</td>
<td>−32768 to 32767</td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
<td>4 bytes</td>
<td>−2147483648 to 2147483647</td>
</tr>
<tr>
<td>long</td>
<td>integer</td>
<td>8 bytes</td>
<td>−9223372036854775808 to 9223372036854775807</td>
</tr>
<tr>
<td>float</td>
<td>floating-point number</td>
<td>4 bytes</td>
<td>−3.40282347 × 10^-38 to −1.40239846 × 10^-38</td>
</tr>
<tr>
<td>double</td>
<td>floating-point number</td>
<td>8 bytes</td>
<td>1.76769313486231570 × 10^-308 to 1.749065635842746544 × 10^-324</td>
</tr>
</tbody>
</table>

Literals

boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
// The number 26, in decimal
int decVal = 26;
// The number 26, in hexadecimal
int hexVal = 0x1a;
// The number 26, in binary
int binVal = 0b11010;
double d1 = 123.4;
float f1 = 123.4f;
Character and String Literals

- \b (backspace),
- \t (tab),
- \n (line feed),
- \f (form feed),
- \r (carriage return),
- \" (double quote),
- \' (single quote),
- \\ (backslash).
- null: used as a value for any reference type (not for primitive types)

Arrays

- An array is a container object that holds a fixed number of values of a single type.

- Declaring a variable to refer to an array
  - int[] anArrayOfChars;
Creating, Initializing and Accessing an Array

// create an array of integers
anArray = new int[10];
anArray[0] = 100; // initialize first element
anArray[1] = 200; // initialize second element
anArray[2] = 300; // and so forth
System.out.println("Element 2 at index 1: "+anArray[1]); // access by index

Alternatively,
int[] anArray = {
    100, 200, 300,
    400, 500, 600,
    700, 800, 900, 1000
};

Multidimensional Array

- An array whose components are themselves arrays (rows are allowed to vary in length)

class MultiDimArrayDemo {
    public static void main(String[] args) {
        String[][] names = {
            {"Mr. ", "Mrs. ", "Ms. "},
            {"Smith", "Jones"}
       );
        // Mr. Smith
        System.out.println(names[0][0] + names[1][0]);
        // Ms. Jones
        System.out.println(names[0][2] + names[1][1]);
    }
}
Copying Arrays

- The `System` class has an `arraycopy()` method to efficiently copy data from one array into another:

```java
public static void arraycopy(Object src, int srcPos,
Object dest, int destPos, int length)
```

class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}

Operators

**Simple Assignment Operator**

=       Simple assignment operator

**Arithmetic Operators**

+       Additive operator (also used for String concatenation)
-       Subtraction operator
*       Multiplication operator
/       Division operator
%       Remainder operator

**Unary Operators**

+       Unary plus operator; indicates positive value
(numbers are positive without this, however)
-       Unary minus operator; negates an expression
++      Increment operator; increments a value by 1
--      Decrement operator; decrements a value by 1
!       Logical complement operator; inverts the value of a boolean
Operators

Equality and Relational Operators

- == Equal to
- != Not equal to
- > Greater than
- >= Greater than or equal to
- < Less than
- <= Less than or equal to

Conditional Operators

- && Conditional-AND
- || Conditional-OR
- ?: Ternary (shorthand for if-then-else statement)

Type Comparison Operator

- instanceof Compares an object to a specified type

Bitwise and Bit Shift Operators

- ~ Unary bitwise complement
- << Signed left shift
- >> Signed right shift
- >>> Unsigned right shift
- & Bitwise AND
- ^ Bitwise exclusive OR
- | Bitwise inclusive OR

instanceOfDemo

class InstanceofDemo {
    public static void main(String[] args) {
        Animal obj1 = new Animal();
        Animal obj2 = new Cat();

        System.out.println("obj1 instanceof Animal: "+(obj1 instanceof Animal));
        System.out.println("obj1 instanceof Cat: "+(obj1 instanceof Cat));
        System.out.println("obj2 instanceof Animal: "+(obj2 instanceof Animal));
        System.out.println("obj2 instanceof Cat: "+(obj2 instanceof Cat));
    }
}

class Animal {}

class Cat extends Animal {}
Pre-, Post-Increment Operator

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

Expressions

• An expression is a construct made up of variables, operators, and method invocations
  o constructed according to the syntax of the language
  o evaluates to a single value

int gear = 0;
anArray[0] = 100;
System.out.println("Element 1 at index 0: " + anArray[0]);
int result = 1 + 2; // result is now 3
if (value1 == value2)
    System.out.println("value1 == value2");
Compound Expressions

- Compound expressions can be constructed from various smaller expressions as long as the data type required by one part of the expression matches the data type of the other.

- $x + y / 100$ // ambiguous
- $(x + y) / 100$ // unambiguous, recommended
- $x + (y / 100)$ // unambiguous, recommended

Operator Precedence

<table>
<thead>
<tr>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>postfix</td>
<td><code>expr++ expr--</code></td>
</tr>
<tr>
<td>unary</td>
<td><code>++expr --expr +expr -expr !</code></td>
</tr>
<tr>
<td>multiplicative</td>
<td><code>* / \</code></td>
</tr>
<tr>
<td>additive</td>
<td><code>+ -</code></td>
</tr>
<tr>
<td>shift</td>
<td><code>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</code></td>
</tr>
<tr>
<td>relational</td>
<td><code>&lt; &gt; &lt;= &gt;= instanceof</code></td>
</tr>
<tr>
<td>equality</td>
<td><code>== !=</code></td>
</tr>
<tr>
<td>bitwise AND</td>
<td><code>&amp;</code></td>
</tr>
<tr>
<td>bitwise exclusive OR</td>
<td><code>-</code></td>
</tr>
<tr>
<td>bitwise inclusive OR</td>
<td>`</td>
</tr>
<tr>
<td>logical AND</td>
<td><code>&amp;&amp;</code></td>
</tr>
<tr>
<td>logical OR</td>
<td>`</td>
</tr>
<tr>
<td>ternary</td>
<td><code>? :</code></td>
</tr>
<tr>
<td>assignment</td>
<td>`= += -= *= /= %= &amp;= ^=</td>
</tr>
</tbody>
</table>
Practice: Evaluating Expressions

Given integer variables a, b, c, d, and e, where a = 1, b = 2, c = 3, d = 4,
evaluate the following expressions:

a + b - c + d
a * b / c
1 + a * b % c
a + d % b - c
e = b = d + c / b - a

Practice: Evaluating Expressions

int answer, value = 4 ;

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. value = value + 1 ;</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2. value++ ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. ++value ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. answer = 2 * value++ ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. answer = ++value / 2 ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. value-- ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. --value ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. answer = --value * 2 ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. answer = value-- / 3 ;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Statements

- Statements are roughly equivalent to sentences in natural languages.
- A statement forms a complete unit of execution.
- Expression statements: terminated with a semicolon (;).
  - Assignment expressions: `aValue = 8933.234;`
  - Any use of `++` or `--`: `aValue++;`
  - Method invocations:
    - `System.out.println("Hello World!");`
  - Object creation expressions:
    - `Bicycle myBike = new Bicycle();`
- Declaration statements: `double aValue = 8933.234;`
- Control flow statements: more later…

Blocks

- A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed.

```java
class BlockDemo {
    public static void main(String[] args) {
        boolean condition = true;
        if (condition) { // begin block 1
            System.out.println("Condition is true.");
        } // end block 1
        else { // begin block 2
            System.out.println("Condition is false.");
        } // end block 2
    }
}
```
Control Flow Statements: if-then-else

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

```java
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);
    }
}
```

Control Flow: switch

```java
public class SwitchDemo {
    public static void main(String[] args) {
        int month = 8;
        String monthString;
        switch (month) {
            case 1: monthString = "January"; break;
            case 2: monthString = "February"; break;
            case 3: monthString = "March"; break;
            case 4: monthString = "April"; break;
            case 5: monthString = "May"; break;
            case 6: monthString = "June"; break;
            case 7: monthString = "July"; break;
            case 8: monthString = "August"; break;
            case 9: monthString = "September"; break;
            case 10: monthString = "October"; break;
            case 11: monthString = "November"; break;
            case 12: monthString = "December"; break;
            default: monthString = "Invalid month"; break;
        }
        System.out.println(monthString);
    }
}
```
public class SwitchDemoFallThrough {
    public static void main(String[] args) {
        java.util.ArrayList<String> futureMonths = new java.util.ArrayList<String>();
        int month = 8;
        switch (month) {
            case 1: futureMonths.add("January"); case 2: futureMonths.add("February");
            case 3: futureMonths.add("March"); case 4: futureMonths.add("April");
            case 5: futureMonths.add("May"); case 6: futureMonths.add("June");
            case 7: futureMonths.add("July"); case 8: futureMonths.add("August");
            case 9: futureMonths.add("September"); case 10: futureMonths.add("October");
            case 11: futureMonths.add("November");
            case 12: futureMonths.add("December"); break;
            default: break; }
        if (futureMonths.isEmpty()) {
            System.out.println("Invalid month number");
        } else {
            for (String monthName : futureMonths) {
                System.out.println(monthName);
            }
        }
    }
}

Notes on Switch

• A switch works with the byte, short, char, and int primitive data types. It also works with enumerated types (discussed in Enum Types), the String class, and a few special classes that wrap certain primitive types: Character, Byte, Short, and Integer (discussed later...).

• In Java SE 7 and later, you can use a String object in the switch statement's expression.
Control Flow: while, do-while

while (expression) {
    statement(s)
}

do {
    statement(s)
} while (expression);

The statements within the do block are always executed at least once.

---

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

class DoWhileDemo {
    public static void main(String[] args) {
        int count = 1;
        do {
            System.out.println("Count is: " + count);
            count++;
        } while (count < 11);
    }
}
Control Flow: for

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

class EnhancedForDemo {
    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);
        }
    }
}

Control Flow: Branching - break

- Problem:
  - Search for a specific number in an array.
Control Flow: Branching - break

```java
class BreakWithLabelDemo {
    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 " + i);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

- Problem:
  - Search for a value in a two-dimensional array.
Control Flow: Branching - break

```java
class BreakWithLabelDemo {
    public static void main(String[] args) {
        int[][] arrayOfInts = {
            { 32, 87, 3, 589 },
            { 12, 1076, 2000, 8 },
            { 622, 127, 77, 955 } 
        };
        int searchfor = 12;
        int i; int j = 0;
        boolean foundIt = false;
        search:
        for (i = 0; i < arrayOfInts.length; i++) {
            for (j = 0; j < arrayOfInts[i].length; j++) {
                if (arrayOfInts[i][j] == searchfor) {
                    foundIt = true;
                    break search;
                }
            }
        }
        if (foundIt) {
            System.out.println("Found " + searchfor + " at " + i + ", " + j);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

Control Flow: continue

- Problem:
  - Count the number of occurrences of a specific character in a given string.
Control Flow: continue

The continue statement skips the current iteration of a for, while, or do-while loop.

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.");
    }
}

• Problem:
  • Search for a substring within another string.
class ContinueWithLabelDemo {
    public static void main(String[] args) {
        String searchMe = "Look for a substring in me";
        String substring = "sub";
        boolean foundIt = false;
        int max = searchMe.length() - substring.length();
        test:
            for (int i = 0; i <= max; i++) {
                int n = substring.length(); int j = i; int k = 0;
                while (n-- != 0) {
                    if (searchMe.charAt(j++) != substring.charAt(k++)) {
                        continue test;
                    }
                }
                foundIt = true; break test;
            }
        System.out.println(foundIt ? "Found it" : "Didn't find it");
    }
}

Control Flow: return

• The return statement exits from the current method, and control flow returns to where the method was invoked.
• Two forms:
  o returns a value, e.g., return ++count;
  o doesn't return a value (when a method is declared void), e.g., return;