Introduction to Java

Selim Aksoy
Bilkent University
Department of Computer Engineering
saksoy@cs.bilkent.edu.tr

Java
- A programming language specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid program statements
- The Java programming language was created by Sun Microsystems, Inc.
- It was introduced in 1995 and its popularity has grown quickly since
- It is an object-oriented language

Introduction to Objects
- An object represents something with which we can interact in a program
- An object provides a collection of services that we can tell it to perform for us
- The services are defined by methods in a class that defines the object
- A class represents a concept, and an object represents the embodiment of a class
- A class can be used to create multiple objects

Objects and Classes
- A class (the concept)
- An object (the realization)

Objects from the same class
- John’s Bank Account
  - Balance: $5,257
- Bill’s Bank Account
  - Balance: $1,245,069
- Mary’s Bank Account
  - Balance: $16,833

Inheritance
- One class can be used to derive another via inheritance
- Classes can be organized into inheritance hierarchies

Abstraction
- An abstraction hides (or suppresses) the right details at the right time
- An object is abstract in that we do not have to think about its internal details in order to use it
- If we group information into chunks (such as objects) we can manage many complicated pieces at once
- Classes and objects help us write complex software
- A class is used to model
  - all attributes/properties of an abstraction
  - all behaviors/operations of an abstraction
Encapsulation
- Classes support a particular kind of abstraction: encouraging separation between an object’s operations and their implementations
- Objects are regarded as “black boxes” whose internals are hidden
- Separation of contract (i.e., which operations are available) and implementation of those operations
  - A class can be viewed as a contract; the contract specifies which operations are offered by the class
  - A class can be viewed as an implementation; the implementation specifies how the desired behavior is produced

Java Program Structure
- In the Java programming language:
  - A program is made up of one or more classes
  - A class contains one or more methods
  - A method contains program statements
  - Attributes/properties correspond to fields (or variables)
  - Behaviors/operations correspond to methods
  - A Java application always contains a method called `main`

Java Program Structure
```java
public class MyProgram {
    // comments about the class
}

public static void main(String[] args) {
    // comments about the method
}
```

Comments
- Comments in a program are called inline documentation
- Java comments can take three forms:
  - `//` this comment runs to the end of the line
  - `/*` this symbol runs to the terminating symbol, even across line breaks `*/`
  - `/**` this is a javadoc comment `*/`
Identifiers

- Identifiers are the words a programmer uses in a program.
- An identifier can be made up of letters, digits, the underscore character (_), and the dollar sign ($).
- Identifiers cannot begin with a digit.
- Java is case sensitive—Total, total, and TOTAL are different identifiers.
- By convention, Java programmers use different case styles for different types of identifiers, such as:
  - title case for class names - Lincoln
  - upper case for constants - MAXIMUM

Reserved Words

The Java reserved words:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
<td>else</td>
<td>interface</td>
<td>super</td>
<td></td>
<td></td>
</tr>
<tr>
<td>boolean</td>
<td>extends</td>
<td>long</td>
<td>switch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>break</td>
<td>false</td>
<td>native</td>
<td>synchronized</td>
<td></td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>final</td>
<td>new</td>
<td>this</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case</td>
<td>finally</td>
<td>null</td>
<td>throw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>catch</td>
<td>float</td>
<td>package</td>
<td>transient</td>
<td></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>goto</td>
<td>protected</td>
<td>true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>if</td>
<td>public</td>
<td>try</td>
<td></td>
<td></td>
</tr>
<tr>
<td>const</td>
<td>implements</td>
<td>return</td>
<td>void</td>
<td></td>
<td></td>
</tr>
<tr>
<td>continue</td>
<td>import</td>
<td>short</td>
<td>volatile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>interface</td>
<td>static</td>
<td>while</td>
<td></td>
<td></td>
</tr>
<tr>
<td>do</td>
<td>int</td>
<td>strictfp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>double</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

White Space

- Spaces, blank lines, and tabs are called white space.
- White space is used to separate words and symbols in a program.
- Extra white space is ignored.
- A valid Java program can be formatted in many ways.
- Programs should be formatted to enhance readability, using consistent indentation.

Poorly Formatted Example

```java
//************************************************************
//  Lincoln2.java       Author: Lewis/Loftus
//  Demonstrates a poorly formatted, though valid, program.
//************************************************************
public class Lincoln2{
    public static void main(String[]args){
        System.out.println("A quote by Abraham Lincoln:");
        System.out.println("Whatever you are, be a good one.");
    }
}
```

```java
//********************************************************************
//  Lincoln3.java       Author: Lewis/Loftus
//  Demonstrates another valid program that is poorly formatted.
//********************************************************************
public class Lincoln3{
    public static void main(String[] args){
        System.out.println("A quote by Abraham Lincoln:");
        System.out.println("Whatever you are, be a good one.");
    }
}
```
**Java Translation**

- The Java compiler translates Java source code into a special representation called bytecode.
- Java bytecode is not the machine language for any traditional CPU.
- Another software tool, called an interpreter, translates bytecode into machine language and executes it.
- Therefore, the Java compiler is not tied to any particular machine.
- Java is considered to be architecture-neutral.

**Using Objects**

- The `System.out` object represents a destination to which we can send output.
- In the Lincoln program, we invoked the `println` method of the `System.out` object:

  ```java
  System.out.println("Whatever you are, be a good one.");
  
  object method information provided to the method (parameters)
  ```

- The `System.out` object also provides the `print` method that is similar to the `println` method, except that it does not advance to the next line.

**Character Strings**

- Every character string is an object in Java, defined by the `String` class.
- Every string literal, delimited by double quotation marks, represents a `String` object.
- The string concatenation operator (+) is used to append one string to the end of another.
- It can also be used to append a number to a string.
- A string literal cannot be broken across two lines in a program.

**Example**

```java
// Facts.java Author: Lewis/Loftus
// Demonstrates the use of the string concatenation operator and the automatic conversion of an integer to a string.

public class Facts {
    // Prints various facts.
    public static void main(String[] args) {
        // Strings can be concatenated into one long string
        System.out.println("We present the following facts for your extracurricular edification:");
        System.out.println();
        // A string can contain numeric digits
        System.out.println("Letters in the Hawaiian alphabet: 12");
        // A numeric value can be concatenated to a string
        System.out.println("Dialing code for Antarctica: 672");
        System.out.println("Year in which Leonardo da Vinci invented the parachute: 1515");
        System.out.println("Speed of ketchup: " + 40 + " km per year");
    }
}
```

**String Concatenation**

- The plus operator (+) is also used for arithmetic addition.
- The function that the + operator performs depends on the type of the information on which it operates.
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation.
- If both operands are numeric, it adds them.
- The + operator is evaluated left to right.
- Parentheses can be used to force the operation order.
Example

```java
//********************************************************************
//  Addition.java Author: Lewis/Loftus
//
//  Demonstrates the difference between the addition and string
//  concatenation operators.
//********************************************************************
public class Addition
{
  //-----------------------------------------------------------------
  //  Concatenates and adds two numbers and prints the results.
  //-----------------------------------------------------------------
  public static void main (String[] args)
  {
    System.out.println("24 and 45 concatenated: " + 24 + 45);
    System.out.println("24 and 45 added: " + (24 + 45));
  }
}
```

Escape Sequences

- What if we wanted to print a double quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string:
  ```java
  System.out.println ("I said "Hello" to you.");
  ```
- An escape sequence is a series of characters that represents a special character.
- An escape sequence begins with a backslash character (`\`), which indicates that the character(s) that follow should be treated in a special way:
  ```java
  System.out.println ("I said \"Hello\" to you.");
  ```

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>backspace</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\n</td>
<td>newline</td>
</tr>
<tr>
<td>\r</td>
<td>carriage return</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
</tbody>
</table>

Variables

- A variable is a name for a location in memory.
- A variable must be declared by specifying the variable's name and the type of information that it will hold:
  ```java
  int total;
  int count, temp, result;
  ```
- Multiple variables can be created in one declaration.

Assignment

- An assignment statement changes the value of a variable.
- The assignment operator is the `=` sign:
  ```java
  total = 55;
  ```
- The expression on the right is evaluated and the result is stored in the variable on the left.
- The value that was in `total` is overwritten.
- You can assign only a value to a variable that is consistent with the variable's declared type.
Example

```java
public class Geometry {
    public static void main(String[] args) {
        int sides = 7; // declaration with initialization
        System.out.println("A heptagon has " + sides + " sides.");
        sides = 10;  // assignment statement
        System.out.println("A decagon has " + sides + " sides.");
        sides = 12;
        System.out.println("A dodecagon has " + sides + " sides.");
    }
}
```

Constants

- A constant is an identifier that is similar to a variable except that it holds one value while the program is active.
- The compiler will issue an error if you try to change the value of a constant during execution.
- In Java, we use the `final` modifier to declare a constant.
  ```java
  final int MIN_HEIGHT = 69;
  ```
- Constants:
  - Give names to otherwise unclear literal values
  - Facilitate updates of values used throughout a program
  - Prevent inadvertent attempts to change a value

Primitive Data

- There are exactly eight primitive data types in Java.
- Four of them represent integers:
  - `byte`
  - `short`
  - `int`
  - `long`
- Two of them represent floating point numbers:
  - `float`
  - `double`
- One of them represents characters:
  - `char`
- And one of them represents boolean values:
  - `boolean`

Numeric Primitive Data

The difference between the various numeric primitive types is their size, and therefore the values they can store:

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>±9.0 x 10^38</td>
<td>±9.0 x 10^38</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>±3.4 x 10^38</td>
<td>±3.4 x 10^38</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>±1.7 x 10^308</td>
<td>±1.7 x 10^308</td>
</tr>
</tbody>
</table>

Characters

- A `char` variable stores a single character from the Unicode character set.
- A character set is an ordered list of characters, and each character corresponds to a unique number.
- The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters.
- It is an international character set, containing symbols and characters from many world languages.
- Character literals are delimited by single quotes:
  ```
  'a' 'b' 'c' 'd' 'e' ...
  ```

Characters

- The ASCII character set is older and smaller than Unicode, but is still quite popular.
- The ASCII characters are a subset of the Unicode character set, including:
  - Uppercase letters: `A, B, C, ...`
  - Lowercase letters: `a, b, c, ...`
  - Punctuation: `period, semi-colon, ...`
  - Digits: `0, 1, 2, ...`
  - Special symbols: `&, |, \, ...`
  - Control characters: `carriage return, tab, ...`
Boolean

- A boolean value represents a true or false condition
- A boolean can also be used to represent any two states, such as a light bulb being on or off
- The reserved words true and false are the only valid values for a boolean type

```java
boolean done = false;
```

Arithmetic Expressions

- An expression is a combination of one or more operands and their operators
- Arithmetic expressions use the operators:
  - Addition +
  - Subtraction -
  - Multiplication *
  - Division / (no ^ operator)
- If either or both operands associated with an arithmetic operator are floating point, the result is a floating point

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

```java
14 / 3  equals?  4
8 / 12  equals?  0
```

- The remainder operator (%) returns the remainder after dividing the second operand into the first

```java
14 % 3  equals?  2
8 % 12  equals?  8
```

Operator Precedence

- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation
- Examples:

```java
a + b + c + d + e
a + b * c - d / e
a / (b + c) - d % e
a / (b * (c + (d - e)))
```

Data Conversions

- Sometimes it is convenient to convert data from one type to another
- For example, we may want to treat an integer as a floating point value during a computation
- Conversions must be handled carefully to avoid losing information
- Widening conversions are safest because they tend to go from a small data type to a larger one (such as a short to an int)
- Narrowing conversions can lose information because they tend to go from a large data type to a smaller one (such as an int to a short)
Data Conversions

- Casting is the most powerful, and dangerous, technique for conversion
  - Both widening and narrowing conversions can be accomplished by explicitly casting a value
  - To cast, the type is put in parentheses in front of the value being converted
  - For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:
    \[
    \text{result} = (\text{float}) \text{total} / \text{count};
    \]

Creating Objects

- A variable holds either a primitive type or a reference to an object
- A class name can be used as a type to declare an object reference variable
  \[
  \text{String title;}
  \]
- No object is created with this declaration
- An object reference variable holds the address of an object
- The object itself must be created separately

Creating Objects

- Generally, we use the `new` operator to create an object
  \[
  \text{title} = \text{new String} \left(\text{"Java Software Solutions"} \right);
  \]
  This calls the String constructor, which is a special method that sets up the object
- Creating an object is called instantiation
- An object is an instance of a particular class

String Methods

- The `String` class has several methods that are useful for manipulating strings
- Many of the methods return a value, such as an integer or a new `String` object
- See the list of `String` methods in the Java API

Example

// Construct different strings
\[
\text{String phrase} = \text{new String} \left(\text{"Change is inevitable"} \right);
\text{String mutation1, mutation2, mutation3, mutation4};
\text{System.out.println \left(\text{"Original string: "} + phrase + \text{\"\"} \right);}
\text{mutation1} = \text{phrase.concat} \left(\text{", except from vending machines.\"} \right);
\text{mutation2} = \text{mutation1.toUpperCase();}
\text{mutation3} = \text{mutation2.replace} \left(\text{\"E\"}, \text{\"X\"};\right)
\text{mutation4} = \text{mutation3.substring} \left(3, 30\right);
\]
// Print each mutated string
\[
\text{System.out.println} \left(\text{"Mutation #1: "} + mutation1 \right);
\text{System.out.println} \left(\text{"Mutation #2: "} + mutation2 \right);
\text{System.out.println} \left(\text{"Mutation #3: "} + mutation3 \right);
\text{System.out.println} \left(\text{"Mutation #4: "} + mutation4 \right);
\]
\[
\text{System.out.println} \left(\text{"Mutated length: "} + mutation4.length() \right);
\]
Class Libraries

- A class library is a collection of classes that we can use when developing programs.
- The Java standard class library is part of any Java development environment.
- Its classes are not part of the Java language per se, but we rely on them heavily.
- The System class and the String class are part of the Java standard class library.
- Other class libraries can be obtained through third party vendors, or you can create them yourself.

Packages

- The classes of the Java standard class library are organized into packages.
- Some of the packages in the standard class library are:

<table>
<thead>
<tr>
<th>Package</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang</td>
<td>General support</td>
</tr>
<tr>
<td>java.applet</td>
<td>Creating applets for the web</td>
</tr>
<tr>
<td>java.awt</td>
<td>Graphics and graphical user interfaces</td>
</tr>
<tr>
<td>java.net</td>
<td>Network communication</td>
</tr>
<tr>
<td>java.util</td>
<td>Utilities</td>
</tr>
<tr>
<td>javax.xml.parsers</td>
<td>XML document processing</td>
</tr>
</tbody>
</table>

The import Declaration

- When you want to use a class from a package, you could use its fully qualified name.
  ```java
  java.util.Random
  ```
- Or you can import the class, and then use just the class name.
  ```java
  import java.util.Random;
  ```
- To import all classes in a particular package, you can use the wildcard character.
  ```java
  import java.util.*;
  ```

Example

```java
import java.util.Random;
public class RandomNumbers {
    public static void main (String[] args) {
        Random generator = new Random();
        int num1;
        float num2;
        num1 = generator.nextInt();
        System.out.println ("A random integer: " + num1);
        num1 = generator.nextInt(10);
        System.out.println ("From 0 to 9: " + num1);
        num1 = generator.nextInt(10) + 1;
        System.out.println ("From 1 to 10: " + num1);
        num1 = generator.nextInt(15) + 20;
        System.out.println ("From 20 to 34: " + num1);
        num1 = generator.nextInt(20) - 10;
        System.out.println ("From -10 to 9: " + num1);
        num2 = generator.nextFloat();
        System.out.println ("A random float [between 0-1]: " + num2);
        num2 = generator.nextFloat() * 6;  // 0.0 to 5.999999
        num1 = (int) num2 + 1;
        System.out.println ("From 1 to 6: " + num1);
    }
}
```

Class Methods

- Some methods can be invoked through the class name, instead of through an object of the class.
- These methods are called class methods or static methods.
- The Math class contains many static methods, providing various mathematical functions, such as absolute value, trigonometry functions, square root, etc.
```
temp = Math.cos(90) + Math.sqrt(delta);
```
The Keyboard Class

- The Keyboard class is NOT part of the Java standard class library.
- It is provided by the authors of the textbook to make reading input from the keyboard easy.
- The Keyboard class is part of a package called cs1.
- It contains several static methods for reading particular types of data.

Example

```java
import cs1.Keyboard;

public class Quadratic {
    //-----------------------------------------------------------------
    //  Determines the roots of a quadratic equation.
    //-----------------------------------------------------------------
    public static void main(String[] args) {
        int a, b, c;  // ax^2 + bx + c
        System.out.print("Enter the coefficient of x squared: ");
        a = Keyboard.readInt();
        System.out.print("Enter the coefficient of x: ");
        b = Keyboard.readInt();
        System.out.print("Enter the constant: ");
        c = Keyboard.readInt();
        // Use the quadratic formula to compute the roots.
        // Assumes a positive discriminant.
        double discriminant = Math.pow(b, 2) - (4 * a * c);
        double root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
        double root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);
        System.out.println("Root #1: "+ root1);
        System.out.println("Root #2: "+ root2);
    }
}
```

Formatting Output

- The NumberFormat class has static methods that return a formatter object.
  - `getCurrencyInstance()`
  - `getPercentInstance()`

  Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format.

Example

```java
import cs1.Keyboard;
import java.text.NumberFormat;

public class Price {
    //-----------------------------------------------------------------
    //  Calculates the final price of a purchased item using values
    //  entered by the user.
    //-----------------------------------------------------------------
    public static void main(String[] args) {
        final double TAX_RATE = 0.06;  // 6% sales tax
        int quantity;
        double subtotal, tax, totalCost, unitPrice;
        System.out.print("Enter the quantity: ");
        quantity = Keyboard.readInt();
        System.out.print("Enter the unit price: ");
        unitPrice = Keyboard.readDouble();
        subtotal = quantity * unitPrice;
        tax = subtotal * TAX_RATE;
        totalCost = subtotal + tax;
        // Print output with appropriate formatting
        NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
        NumberFormat fmt2 = NumberFormat.getPercentInstance();
        System.out.println("Subtotal: "+ fmt1.format(subtotal));
        System.out.println("Tax: "+ fmt1.format(tax) + " at "+ fmt2.format(TAX_RATE));
        System.out.println("Total: "+ fmt1.format(totalCost));
    }
}
```

Formatting Output

- The DecimalFormat class can be used to format a floating point value in generic ways.
  - For example, you can specify that the number should be printed to three decimal places.
  - The constructor of the DecimalFormat class takes a string that represents a pattern for the formatted number.

Example

```java
import cs1.Keyboard;
import java.text.DecimalFormat;

public class CircleStats {
    //-----------------------------------------------------------------
    //  Calculates the area and circumference of a circle given its
    //  radius.
    //-----------------------------------------------------------------
    public static void main(String[] args) {
        int radius;
        double area, circumference;
        System.out.print("Enter the circle's radius: ");
        radius = Keyboard.readInt();
        area = Math.PI * Math.pow(radius, 2);
        circumference = 2 * Math.PI * radius;
        // Round the output to three decimal places
        DecimalFormat fmt = new DecimalFormat("0.###");
        System.out.println("The circle's area: "+ fmt.format(area));
        System.out.println("The circle's circumference: " + fmt.format(circumference));
    }
}
```