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 it's popularity has grown quickly since
- It is an object-oriented language

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*
- These terms will be explored in detail throughout the course
- A Java application always contains a method called `main`

Java Program Structure

```
//  comments about the class
public class MyProgram
{



}
```

class header

class body

Comments can be placed almost anywhere

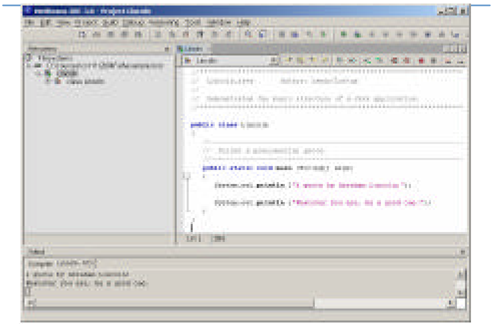
Java Program Structure

```
//  comments about the class
public class MyProgram
{

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

   }

}
```

method header

method body

NetBeans IDE



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`

Identifiers

- Sometimes we choose identifiers ourselves when writing a program (such as `Lincoln`)
- Sometimes we are using another programmer's code, so we use the identifiers that they chose (such as `println`)
- Often we use special identifiers called *reserved words* that already have a predefined meaning in the language
- A reserved word cannot be used in any other way

Reserved Words

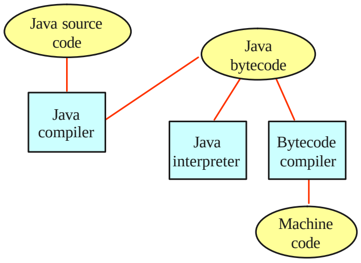
- The Java reserved words:

```
abstract     else         interface    super
boolean      extends      long         switch
break        false        native       synchronized
byte         final        new          this
case         finally      null         throw
catch        float        package      throws
char         for          private      transient
class        goto         protected    true
const        if           public       try
continue     implements   return       void
default      import       short        volatile
do           instanceof   static       while
double       int          strictfp
```

Java Translation

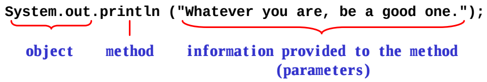
- The Java compiler translates Java source code into a special representation called *bytecode*
- Java bytecode is not the machine language for any traditional CPU
- Another software tool, called an *interpreter*, translates bytecode into machine language and executes it
- Therefore the Java compiler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

Java Translation



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:



```
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

```
//********************************************************************
//  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

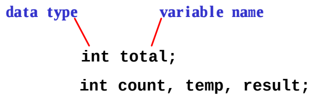
- The plus operator (`+`) is also used for arithmetic addition
- The function that the `+` operator performs depends on the type of the information on which it operates
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation
- If both operands are numeric,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

Example

```
//********************************************************************
//  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));
   }
}
```

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



data type

variable name

```
int total;
```

```
int count, temp, result;
```

Multiple variables can be created in one declaration

Variables

- A variable can be given an initial value in the declaration

```
        int sum = 0;
        int base = 32, max = 149;
```

- When a variable is referenced in a program, its current value is used

Assignment

- An *assignment statement* changes the value of a variable
- The assignment operator is the = sign

```
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

```
//**********************************************************************
//  Geometry.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an assignment statement to change the
//  value stored in a variable.
//**********************************************************************

public class Geometry
{
   //-----------------------------------------------------------------
   //  Prints the number of sides of several geometric shapes.
   //-----------------------------------------------------------------
   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

```
        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 Types

- 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:

Type	Storage	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	< -9 x 10^{18}	> 9 x 10^{18}
float	32 bits	+/- 3.4 x 10^{38} with 7 significant digits	
double	64 bits	+/- 1.7 x 10^{308} with 15 significant digits	

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'   'X'    '7'    '$'    ','    '\n'
```

Boolean

- A `boolean` value represents a true or false condition
- A boolean also can 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

```
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	**/**	
Remainder	**%**	**(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)

14 / 3	equals?	**4**
8 / 12	equals?	**0**

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

14 % 3	equals?	**2**
8 % 12	equals?	**8**

Operator Precedence

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

```
a + b + c + d + e          a + b * c - d / e
  1   2   3   4              3   1   4   2

a / (b + c) - d % e        a / (b * (c + (d - e)))
  2      1    4   3          4     3     2     1
```

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

- In Java, data conversions can occur in three ways:
  - assignment conversion
  - arithmetic promotion
  - casting
- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
  - Only widening conversions can happen via assignment
- *Arithmetic promotion* happens automatically when operators in expressions convert their operands

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`:

```
result = (float) total / 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*

```
String title;
```

- No object is created with this declaration
- The object itself must be created separately

Creating Objects

- Generally, we use the `new` operator to create an object

```
title = new String ("Java Software Solutions");
```

This calls a special method that sets up the object

- An object is an *instance* of a particular class

Creating Objects

- Because strings are so common, we don't have to use the `new` operator to create a `String` object

```
title = "Java Software Solutions";
```

- This is special syntax that works only for strings
- Once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
title.length()
```

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
String phrase = new String ("Change is inevitable");
String mutation1, mutation2, mutation3, mutation4;

System.out.println ("Original string: \"" + phrase + "\"");
System.out.println ("Length of string: " + phrase.length());

mutation1 = phrase.concat (", except from vending machines.");
mutation2 = mutation1.toUpperCase();
mutation3 = mutation2.replace ('E', 'X');
mutation4 = mutation3.substring (3, 30);

// Print each mutated string
System.out.println ("Mutation #1: " + mutation1);
System.out.println ("Mutation #2: " + mutation2);
System.out.println ("Mutation #3: " + mutation3);
System.out.println ("Mutation #4: " + mutation4);

System.out.println ("Mutated length: " + mutation4.length());
```

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:

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities and components
java.net	Network communication
java.util	Utilities
javax.xml.parsers	XML document processing

The import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Random
```

- Or you can *import* the class, and then use just the class name

```
import java.util.Random;
```

- To import all classes in a particular package, you can use the * wildcard character

```
import java.util.*;
```

- The `Random` class is part of the `java.util` package and provides methods that generate pseudorandom numbers

Example

```
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

```
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);
   }
}
```