Java Programming: Guided Learning with Early Objects

Chapter 2 - Input/Output

Objectives

- Use input statements to input data into memory
- Use output statements to output results and how to format output using the method `printf`
- Debug using `printlns`
- Import packages and why packages are necessary
- Deal with file input and output
- Use input and output dialog boxes and become familiar with the `String` method `format`

Input (Read) Statement Using class Scanner

- The statement creates the input stream object console and associates it with the standard input device (keyboard).

- The `Scanner` class puts data into variables from the standard input device

```
static Scanner console = new Scanner (System.in);
```

System.in is called standard input stream objects. It is designed to retrieve the desired input data from the standard input device in a form of byte.

- **The input is:**
  - **Integer** : `console.nextInt()  // used when the input desired is of type  integer.`
  - **Double** : `console.nextDouble()    // used when the input desired  is of type double.`
  - **String** : `console.next()   // used when the input desired is of type string.`
  - **String** : `console.nextLine()  //  retrieves input of`

```
type string until the user hits enter ( until end
of line ).
```

Note : The class Scanner is available in java version 5.0 or higher. It is not available in versions less than 5.0

Variable Initialization

- Variable can be initialized with a literal value (to change, you must edit the code). Suppose that feet is of type integer and it is assigned the value of 35. One way to define this variable is as follows :

```
int feet = 35;
```

- Another way this variable can be initialized with an input statement (More flexible)

```
int feet = console.nextInt();
```

this statement causes the computer to get the input data (35 in our example) from the standard input device (keyboard) and stores it in the variable feet.

Sample Program

```
//  Program to demonstrate how to read string and numeric

import java.util.*;       //  must import java utility library

public class Example
{
static Scanner console = new Scanner(System.in);   //  define the standard
                                                    // input device

   public static void main(String[] args)
   {
      String firstName;
      String lastName;

      int age;
      double weight;

      System.out.println("Enter first name, last "
                  + "name, age, and weight "
                  + "separated by spaces.");

      firstName = console.next();       // retrieve first name
      lastName = console.next();        // retrieve last name
      age = console.nextInt();          // retrieve age
      weight = console.nextDouble();         //  retrieve weight

      System.out.println("Name: " + firstName  + " " + lastName);

      System.out.println("Age: " + age);
      System.out.println("Weight: " + weight);        }            }
```

SAMPLE RUN

```
Enter first name, last name, age, and weight separated by spaces.
John Jimbo 34 69
Name: John Jimbo
Age: 34
Weight: 69.0
```

Reading a Single Character

- Assume next input is a single character
  `char ch = console.next().charAt(0);`
- When input ‘A’, do not include quotations
- When storing a string from an input statement, do not include double quotations
- Expression `charAt(0)` selects first character of a string
- Expression `charAt(5)` selects fifth character of a string
- If program results are incorrect, do a walk-through with statements that assign values to variables

Sample Program

```
import java.util.*;      //  must import java utility library

public class Example
{
static Scanner console = new Scanner(System.in);   //  define the standard
                                                    // input device


   public static void main (String[] args)
   {     char  firstName;
         firstName = console.next().charAt(0);
         System.out.println(firstName);  }     }
```

Sample Output

```
Husain
H
```

Output

- Standard output device: usually the monitor
- Methods accessed by `System.out`:
  - `print`
  - `println`
  - `printf`

Output with `print` and `println`

- Syntax:

```
System.out.print(expression);
System.out.println(expression);
System.out.println();
```

Sample Example

```
public class Main {

   //  Program written By Husain Gholoom to illustrate the print statements

   public static void main(String[] args) {

      int value1 = 43, value2 = 10;
      int sum, difference, product, quotient,modulus;
      boolean isItPayday = false;
      boolean areYouBroke = true;
      boolean sixIsBigger  = ( 6 > 5 );
      boolean sevenIsSmaller  = ( 7 >  9 );

      System.out.println("The sum is   " + (value1 + value2));
      System.out.println("The sum is   " + (45 + 25));
      System.out.println("The product  is  " + (value1 * value2));

      System.out.print("Is It Pay Day   =   ") ;
      System.out.print("Are You Broke    ");

      System.out.println();
      System.out.println("isSixBigger    "  + sixIsBigger);
      System.out.println("sevenIsSmaller   "  + sevenIsSmaller);  }  }
```

Sample Output

```
The sum is   53
The sum is   70
The product  is  430
Is It Pay Day   =   Are You Broke
isSixBigger    true
sevenIsSmaller   false
```

Escape Sequences

- Newline character: \n
  - – Moves insertion point to beginning of next line

For Example

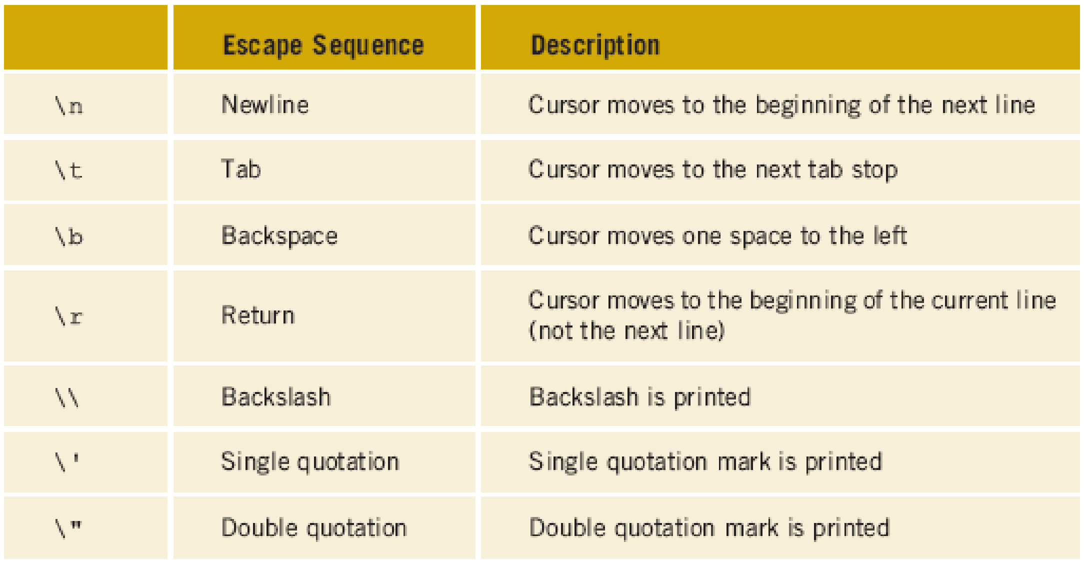
```
System.out.println("Hello World . \n My Name is Billy Bob. \n This is Java Course");
```

The out of the previous statement is

```
Hello World .
My Name is Billy Bob.
This is Java Course
```

Note : System.out.println("\n") is the same as System.out.println();

- Other escape sequences allow control of the output



	Escape Sequence	Description
`\n`	Newline	Cursor moves to the beginning of the next line
`\t`	Tab	Cursor moves to the next tab stop
`\b`	Backspace	Cursor moves one space to the left
`\r`	Return	Cursor moves to the beginning of the current line (not the next line)
`\\`	Backslash	Backslash is printed
`\'`	Single quotation	Single quotation mark is printed
`\"`	Double quotation	Double quotation mark is printed

Sample Program

```
// This program shows the effect of escape sequences.
//*******************************************************

public class Example
{
   public static void main (String[] args)
   {
      System.out.println("The newline escape sequence is \\n");
      System.out.println("Hello World . \nMy Name is Billy Bob. \nThis is Java Course");
      System.out.println("This is how The tab \tworks");
      System.out.println("\n");
      System.out.println("The previous statement was same as System.out.println()");
      System.out.println("The tab character is represented as \'\\t\'");
      System.out.println("The tab character is represented as '\\t'");
      System.out.println("The string \"Sunny\" contains five characters");
   }     }
```

Sample output

```
The newline escape sequence is \n
Hello World .
My Name is Billy Bob.
This is Java Course
This is how The tab	works


The previous statement was same as System.out.println()
The tab character is represented as '\t'
The tab character is represented as '\t'
The string "Sunny" contains five characters
```

Debugging: Sprinkling with `printlns`

- Insert temporary `println` statements to discover semantic errors
- After correcting the problem, remove `printlns`

Packages, Classes, Methods, and the `import` Statement

- Package: collection of related classes
  - Every package has a name
- Uses of term **class**:
  - Create Java programs
  - Group a set of related operations
  - Allow users to create data types

- **Method**: set of instructions to accomplish a specific task

- Package `java.lang` contains `class Math`
  - Methods for mathematical operations
- Package `java.util` contains `class Scanner`
  - Methods `nextInt`, `nextDouble`, `next`, `nextLine`
- `import` statement makes package contents available to the program

- Syntax:

```
import packageName.*;
```

- Examples :-

```
import java.lang.*;
import java.util.*;
import java.io.*;
```

- `import` is a reserved word

- Import a specific class from a package:

```
import java.util.Scanner;
```

- Import statements placed at top of the program

Formatting Output with printf

- Methods `print` and `println` cannot format output directly
  - – Example: default precision of floats is six decimal places, double is 15
- Method `printf` formats output in a specific way
- Syntax:

```
    System.out.printf(formatString); or
    System.out.printf(formatString, argumentList);
```

- Example:

Assuming that the value of centimeters is 150, what the output of the two following statements

```
  System.out.println("There are "+ (150/2.54)+" inches in "+
  centimeters + " centimeters");
```

and

```
  System.out.printf("There are %.2f     inches in %d
  centimeters.%n",  centimeters / 2.54, centimeters);
```

- In the example, `%.2f` and %d are format specifiers

The output of the above statements are :-

The first statement

```
There are 59.05511811023622  inches in 150 centimeters
```

And the second one is

```
There are 59.06        inches in 150 centimeters.
```

Sample Program

```
// Program illustrates how to format the outputting of
// decimal numbers.

public class FormattingDecimalNum
{
    public static void main(String[] args)
    {
        double x = 15.674;
        double y = 235.73;
        double z = 9525.9864;
        int num  =  123456; String Str = “PAAET”
        System.out.println("1-   The values of x, "
                         + "y, and z with two "
                         + "decimal places.");
        System.out.printf("2-   x = %.2f %n", x);
        System.out.printf("3-   y = %.2f %n", y);
        System.out.printf("4-   z = %.2f %n", z);

        System.out.println("5-  The values of x, "
                         + "y, and z with three "
                         + "decimal places.");
        System.out.printf("6-  x = %6.3f %n", x);
        System.out.printf("7-  y = %6.3f %n", y);
        System.out.printf("8-  z = %1.1f %n", z);
        System.out.printf("9-  z = %10d %6.2f %n", num,x);
        System.out.printf("10- Str = %10s%n", Str);
        System.out.printf("%2d%7.2f%15s%n",num, x, Str);
        System.out.printf("%15s%6d%9.2f%n",Str, num, x);
        System.out.printf("%8.2f%7d%15s%n",x, num, Str); }  }
```

Sample Run

```
1-   The values of x, y, and z with two decimal places.
2-   x = 15.67
3-   y = 235.73
4-   z = 9525.99
5-  The values of x, y, and z with three decimal places.
6-  x = 15.674
7-  y = 235.730
8-  z = 9526.0

9-  z =     123456   15.67
10- Str =      PAAET

123456  15.67          PAAET
          PAAET123456    15.67
   15.67 123456          PAAET
```

Using Dialog Boxes for Input/Output

- Program performs three basic operations:
  - Gets data into the program
  - Manipulates data
  - Outputs the results
- Input data using `class Scanner`
- Display output with `print`, `println`, `printf`
- Input to program using dialog boxes is in string format
- Numeric strings must be converted to numbers

Parsing Numeric Strings

- **Numeric string**: string consisting of only integer or floating-point number
  - – May be preceded by a minus sign

- Must be converted to numeric form to use in mathematical operations
  - – `Integer.parseInt(str)` converts to `int`
  - – `Float.parseFloat(str)` converts to `float`
  - – `Double.parseDouble(str)` converts to `double`

- **Wrapper classes contain methods to convert numeric strings into numbers**

- `Integer`, `Float`, and `Double` are wrapper classes

Sample Program

```
/ *
 * @author Husain Gholoom
 */

public class Main {
      int numOne;
      double numTwo, Result;
      float numThree;

      numOne = Integer.parseInt("12");
      numTwo = Double.parseDouble("34");
      numThree = Float.parseFloat("56");
      Result = numOne + numTwo + numThree;
      System.out.println("Parse Integer =  "+numOne);
      System.out.println("Parse Double  =  "+numTwo);
      System.out.println("Parse Floate  =  "+numThree);
      System.out.println("Result in Float    =  "+Result);     }    }
```

Sample Run

```
Parse Integer =  12
Parse Double  =  34.0
Parse Floate  =  56.0
Result in Float    =  102.0
```

Using Dialog Boxes for Input/Output

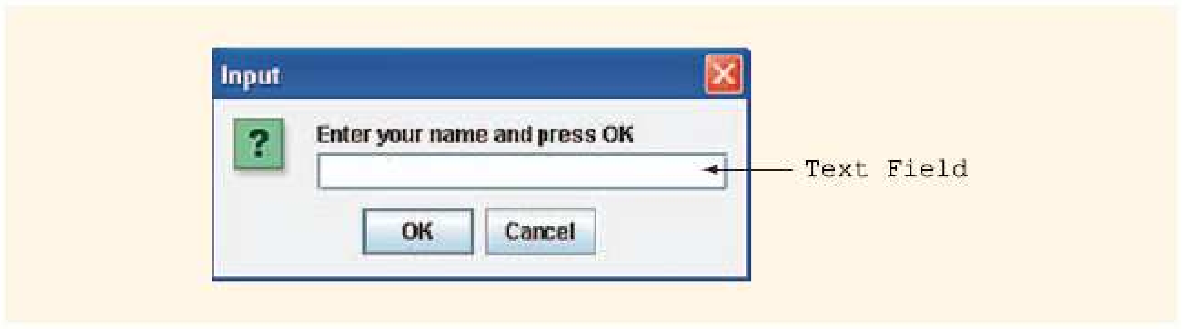
- `class JOptionPane` allows programmer to use GUI components for I/O
- `JOptionPane` contained in package `javax.swing`
- Two methods:
  - `showInputDialog`
    - Allows user to enter String from keyboard
  - `showMessageDialog`
    - Allows programmer to display results

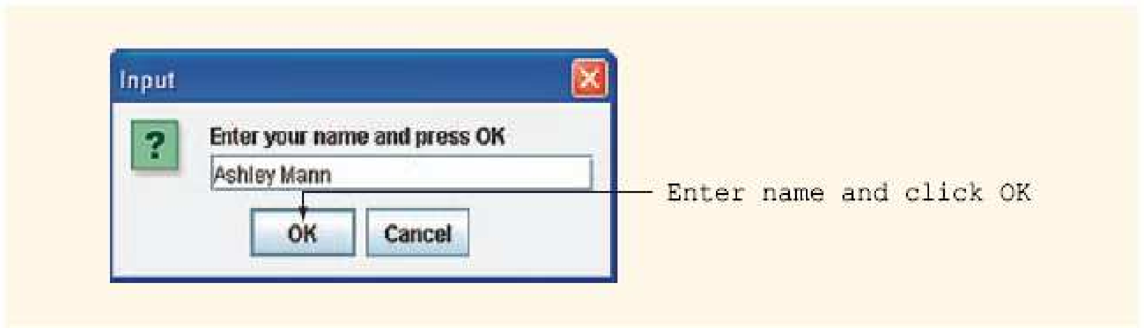
- Syntax for `showInputDialog` :

```
str = JOptionPane.showInputDialog(expression);
```

- Dialog box containing `expression` appears on screen
  - Prompts user for data
  - Data entered returned as a string

- **Text field**: white area where users enter data





- Syntax for `showMessageDialog` :

```
JOptionPane.showMessageDialog(
              parentComponent,
              message StringExpression,
              boxTitleString,
              messageType);
```

Parameter	Description
`parentComponent`	This is an object that represents the parent of the dialog box. For now, we will specify the `parentComponent` to be `null`, in which case the program uses a default component that causes the dialog box to appear in the middle of the screen. Note that `null` is a reserved word in Java.
`messageStringExpression`	The `messageStringExpression` is evaluated and its value appears in the dialog box.
`boxTitleString`	The `boxTitleString` represents the title of the dialog box.
`messageType`	An `int` value represents the type of icon that will appear in the dialog box. Alternatively, you can use certain `JOptionPane` options described below.

`messageType`	Description
`JOptionPane.ERROR_MESSAGE`	The error icon, ❌, is displayed in the dialog box.
`JOptionPane.INFORMATION_MESSAGE`	The information icon, ℹ, is displayed in the dialog box.
`JOptionPane.PLAIN_MESSAGE`	No icon appears in the dialog box.
`JOptionPane.QUESTION_MESSAGE`	The question icon, ❓, is displayed in the dialog box.
`JOptionPane.WARNING_MESSAGE`	The warning icon, ⚠, is displayed in the dialog box.

- `JOptionPane` class contained in package `javax.swing`
- Must import `javax.swing`:

```
  import javax.swing.JOptionPane;
```

or

```
  import javax.swing.*;
```

- In order to use input/output dialog boxes and terminate program execution use:

```
System.exit(0);
```

- Only needed for programs that have GUI components
- If omitted, program does not end when dialog box disappears

Sample Program :-

```
import javax.swing.JOptionPane;

public class DialogBox
{
  public static void main(String[] args)
  {
    String name;
    String str;
    int num1 = 45;
    int num2 = 56;
    int sum;

    name =
      JOptionPane.showInputDialog("Enter your name and press OK");

    JOptionPane.showMessageDialog(null, "Hi  "+name,
                                  "Greetings",
                                 JOptionPane.INFORMATION_MESSAGE);
    JOptionPane.showMessageDialog(null, "You have to pay  = $" + 500.45,
                                  "Invoice",
                                  JOptionPane.PLAIN_MESSAGE);

    str = "The two numbers are: " + num1 + " and " + num2 + "\n";
    sum = num1 + num2;

    str = str + "The sum of the numbers is: " + sum  + "\n";
    str =  str + "That is all for now!";

    JOptionPane.showMessageDialog(null, str, "Summing Numbers",
                                  JOptionPane.ERROR_MESSAGE);
    System.exit(0);
  }
}
```

Formatting the Output Using the `String` Method `format`

- Method `printf` formats output for standard output device
- `printf` cannot be used with output dialog boxes
- Formatting for output dialog box done with `String` method `format`
- Syntax:
```
String.format(formatStr, argList);
```

File Input/Output

- Getting input from keyboard and sending output to monitor has limitations
  – If input data is large, it is inefficient to type it at the keyboard each time program is run
  – Typing is error prone
- Advantages of using a file as input:
  – Prepare the data before running the program
  – Program accesses the data each time it runs
  – Saving output to file allows the output to be reused
- **File**: area in secondary storage used to hold information
  - `Scanner` can read from input sources other than standard input
  - `class FileReader`:
```
Scanner inFile = new Scanner (
                 new FileReader(“prog.dat”));
```
  - Send output to a file using `class PrintWriter`

- File I/O is a four-step process:
  - Import necessary classes
  - Create, associate objects with I/O sources
  - Use methods associated with variables to input/output data
  - Close the files

- Import necessary classes from `java.util` and `java.io`:

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

- Close input and output files using method `close`

Storing (Writing) Output to a File

- `class PrintWriter` stores output to file
- Create `PrintWriter` variable
  - Associate variable with the destination
    - Destination is the file where output stored

- Methods `print`, `println`, `printf` used with `PrintWriter` variable

- Close input and output files using method `close`
  - Ensures the buffer holding the output will be emptied

- Input file must exist before program executes

- Runtime exceptions if files cannot be found or read

- If input file does not exist:
  - Statement associating object with input file fails
  - Throws a `FileNotFoundException`
  - Method `main` also throws the exception

- Output file need not exist before it is opened

- If output file does not exist, computer creates it

- If output file already exists, contents erased when opened by the program

- If program cannot create or access output file, throws `FileNotFoundException`

```
 // Program to illustrate the I/O file usage

import java.io.*;
import java.util.*;

public class Main {

   public static void main(String[] args) throws FileNotFoundException
   {
      //declare and initialize the variables
      double test1, test2, test3, test4, test5;
      double average;
      String firstName;
      String lastName;

      Scanner inFile =  new Scanner(new FileReader("test.txt"));

      PrintWriter outFile = new    PrintWriter("testavg.out");

      firstName = inFile.next();
      lastName = inFile.next();

      outFile.println("Student Name: "  + firstName + " " + lastName);

         // retrieve the five test scores

      test1 = inFile.nextDouble();
      test2 = inFile.nextDouble();
      test3 = inFile.nextDouble();
      test4 = inFile.nextDouble();
      test5 = inFile.nextDouble();

      outFile.printf("Test scores: %5.2f %5.2f %5.2f "
                + "%5.2f %5.2f %n", test1, test2, test3, test4, test5);
      average = (test1 + test2 + test3 + test4+ test5) / 5.0;
      outFile.printf("Average test score: %5.2f %n",average);

      inFile.close();
      outFile.close();         }      }
```

Input file name is test.txt

```
Steve Austin 87.50 89 65.75 37 98.50
```

After successful run, the output file (testavg.txt) consist of the following

```
Student Name: Andrew Miller
Test scores:  87.50 89.00 65.75 37.00 98.50
Average test score: 75.55
```

Debugging: Understanding Error Messages

- Java compiler finds syntactic errors
  - Provides messages describing the errors
- Messages do not always describe problem exactly
- Do not always describe all errors at once
- Correcting single error often results in new errors reported
- Find and correct sources of errors as they are reported

Summary

- Java `class Scanner` reads input from the console
- Methods of `class Scanner`:
  - `nextInt()`
  - `nextDouble()`
  - `next()`
  - `nextLine()`
- Variables can be initialized from user input
- Use the `charAt(0)` method to read a single character
- Output to standard output object `System.out`
  - Methods `print`, `println`, `printf`
- Escape sequences control the output
  - Examples: `‘\t’` , `‘\n’` , `‘\b’` , `’\0’`
- Method `printf` formats output
  - Floating-point precision
  - Columns
- Inserting `println` statements in the code facilitates debugging
- Package is a collection of related classes
  - Every package has a name
- Classes create Java programs
  - Group related operations
  - Allow users to create data types
- Method is a set of instructions to complete a task
- Packages must be imported to be accessible to a program
- Prompt lines help the user understand valid input for a program
- File: area in secondary storage used to hold information
- `class PrintWriter` can write output to file
- Data entered in dialog boxes are strings

- Numeric strings must be converted to numbers to be used in arithmetic operations
  - `Integer.parseInt()`
  - `Float.parseFloat()`
  - `Double.parseDouble()`

- Java `class JOptionPane` allows programmer to use GUI components for I/O
  - `showInputDialog`
  - `showMessageDialog`

- `JOptionPane` contained in `javax.swing`
- Method `printf` cannot be used with dialog boxes
- Method `format` in `class String` formats output for use in dialog boxes

Additional Resources

1. Article about `printf` in Java:
   http://sharkysoft.com/archive/printf/docs/javadocs/lava/clib/stdio/doc-files/introduction.htm

2. An article from the Sun Web site about packages:
   http://java.sun.com/docs/books/jls/second_edition/html/packages.doc.html

3. Sun tutorials on exception handling:
   http://java.sun.com/docs/books/tutorial/essential/exceptions/
   http://java.sun.com/docs/books/tutorial/essential/exceptions/throwing.html

Key Terms

- **file** – An area in secondary storage used to hold information.
- **method** – A set of instructions to accomplish a task.
- **numeric string** – A string consisting only of an integer or floating-point number, optionally preceded by a minus sign.
- **output statements –** Statements using `System.out.print` and `System.out.println`.
- **package –** A collection of related classes.
- **parsing numeric strings –** Converting numeric strings into numeric form.
- **prompt lines –** Executable statements that tell the user what to do to interact with the program.
- **standard output object –** The object `System.out`, which outputs to the standard output device.
- **wrapper classes** – Classes that contain methods to convert a numeric string into a number.