

Java Language Essentials

- ❖ Java is **Case Sensitive**
 - ◆ All **Keywords** are lower case
- ❖ White space characters are ignored
 - ◆ Spaces, tabs, new lines
- ❖ Java statements usually end with a semicolon ;
- ❖ Java compound statements use { }
- ❖ Java Language Comments
 - ◆ Single-line **// Comment goes to end of line**
 - ◆ Multi-line comments **/* This is a comment */**
 - ◆ Create a title block at beginning of program to describe program
 - ◆ Describe purpose of unusual code
- ❖ Use descriptive identifiers



Copyright © 2004 R.M. Laurie 1

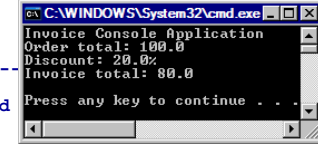
Java Console Application

```

/* Invoice Console Application
 * Author: Robert Laurie
 * Date: 26 March 2004
 * -----
 * Description: This program will
 * calculate the invoice price based
 * on a 20% discount
 */
public class InvoiceConsoleAp
{
    public static void main(String[] args)
    {
        // VARIABLE DECLARATIONS
        double dDiscount, dInvoice, dOrder = 100;

        // PROCESSING
        dDiscount = dOrder * 0.2;
        dInvoice = dOrder - dDiscount;

        // DISPLAY RESULTS
        System.out.println("Invoice Console Application");
        System.out.println("Order total: " + dOrder);
        System.out.println("Discount: " + dDiscount + "%");
        System.out.println("Invoice total: " + dInvoice + "\n");
    }
}
    
```



Java Identifiers

- ❖ Identifier Naming Rules
 - ◆ Identifiers are case sensitive
 - ◆ Begin each identifier with a letter, **_** underscore, or **\$** dollar sign (no numbers)
 - ◆ May use numbers for subsequent characters
 - ◆ No spaces allowed, but may use underscore
 - ◆ Identifiers can be up to 255 characters
 - ◆ Not a Java keyword (approximately 50)
 - ◆ **public class name** must be same as **filename.java**
- ❖ Identifier Naming Guidelines
 - ◆ TitleCase meaningful names (self documenting)
 - ◆ Class name should begin with capital letter
 - ◆ Variable identifiers begin with lower case letter(s) to indicate variable data type

Copyright © 2004 R.M. Laurie 3

Java Applications have Class

- ❖ Java applications are contained in classes
 - ◆ Each **filename.java** file must have one (and only one) **public class** declaration with same name:


```

public class InvoiceConsoleAp
{ // Class definition begins here
    public static void main(String[] args)
    {
        System.out.println("Invoice Application");
    }
} // Class definition ends here
                    
```
- ❖ Every Java applications contain methods
 - ◆ **main()** method is start of program for java
 - ◆ **public** keyword means other classes can access main method
 - ◆ **static** keyword means method can be called from other classes
 - ◆ **void** keyword means method won't return any values
 - ◆ Every main has an argument **args** defined as array of strings

Copyright © 2004 R.M. Laurie 4

Eight Primitive Java Variable Data Types

| Prefix | Type | Byte | Use |
|--------|---------|------|---|
| bi | Byte | 1 | Very short integers from -128 to 127. |
| si | Short | 2 | Short integers from -32,768 to 32,767. |
| i | Int | 4 | Integers from -2,147,483,648 to 2,147,483,647. |
| li | Long | 8 | Long integers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. |
| f | Float | 4 | Single-precision, floating-point numbers from -3.4E38 to 3.4E38 with 6 or 7 significant digits. |
| d | Double | 8 | Double-precision, floating-point numbers from 1E308 to 1E308 with from 14 to 15 significant digits. |
| c | Char | 2 | A single Unicode character that's stored in two bytes. |
| b | Boolean | 1 | A true (1) or false (0) value. |

- ❖ Identifier guideline: Begin variable identifiers with lower case prefix to indicate data type
 - ◆ biTVChan siScore iSSN liPopulation
 - ◆ fAverage dTotalCost
 - ◆ cGrade
 - ◆ bAnswer

Copyright © 2004 R.M. Laurie 5

Java Keywords

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

Copyright © 2004 R.M. Laurie 6

Variable Declaration and Initialization

- ❖ Declaration reserves memory for variable based on data type
 - ◆ double dDiscount;
- ❖ May declare multiple variables of same type by separating with comma
 - ◆ double dDiscount, dInvoice;
 - ◆ int iCount, iScore, iMin;
- ❖ May initialize in declaration statement
 - ◆ double dDiscount=0.2;
 - ◆ int iCount=0, iMaxScore=100;

Copyright © 2004 R.M. Laurie 7

Preferred Data Types and Constants

- ❖ Preferable to use int data type not long
 - ◆ int's 32 bits is usually enough and long not needed
 - ◆ For long initialization must designate with L
 - ◆ long liDiscount=0.2L
- ❖ Preferable to use double data type not float
 - ◆ float's 6 significant digits is usually inadequate
 - ◆ For float initialization must designate with F
 - ◆ float fDiscount=0.2F
- ❖ Constants – Value can't be change by program
 - ◆ Reserves memory for associated data type and assigns a value that can not be changed
 - ◆ Use final keyword to declare as constant
 - ◆ Constant naming convention is UPPER_CASE
 - ◆ final double DAYS_IN_MAY = 31;

Copyright © 2004 R.M. Laurie 8

Arithmetic Operators Precedence

(Highest to Lowest)

| | |
|-------|--|
| () | Defines order of operation |
| - | Negative (unary) |
| * / % | Multiply, Division, Remainder |
| + - | Addition (String Concatenation), Subtraction |
| = | Assignment |

Copyright © 2004 R.M. Laurie 9

Arithmetic Operator Examples

```
double dRemainder, dAvgScore;
int iScore = 93, iScoreCount = 0, iTotalScore;
iTotalScore = iScore;
iScore = 78;
iTotalScore = iTotalScore + iScore;
iScoreCount = iScoreCount + 1;
dAvgScore = iTotalScore / iScoreCount;
dRemainder = iTotalScore % iScoreCount;
System.out.println("F < " + dAvgScore / 2);
```

Copyright © 2004 R.M. Laurie 10

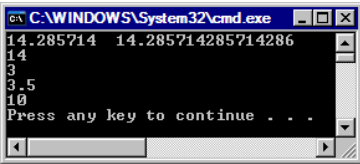
Type Casting

- ❖ **Implicit Type Casting**
 - ◆ Automatically converts less precise to more precise types: int → long → float → double
 - ◆ If one operand is float and other is double the result will be double
- ❖ **Explicit Type Casting**
 - ◆ Overrides implicit type cast
 - ◆ (type) operand

```
double dAverageScore;
int iGradeCourse = (int) dAverageScore;
```
- ❖ **Declaration Type Casting**
 - ◆ Assign results of calculation to variable declaration
 - ◆ `double dAverage = (dA + nB + nC)/3;`
 - ◆ `int iFinalScore = (int) dAverage;`

Copyright © 2004 R.M. Laurie 11

Java Console



```
/* Calculate Console Ap
 * Author: Robert Lauri
 */
public class CalculateConsoleAp
{
    public static void main(String[] args)
    {
        int iA = 20, iB = 70;
        float fC = 100;
        double dD = 100;

        System.out.println(fC/7 + " " + dD/7);
        System.out.println((int)fC/7);
        System.out.println(iB/iA);
        System.out.println((float)iB/iA);
        System.out.println(iB%iA);
    }
}
```

Copyright © 2004 R.M. Laurie 12