Java Language Essentials

- Java is Case Sensitive
  - All Keywords are lower case
- White space characters are ignored
  - Spaces, tabs, new lines
- Java statements must end with a semicolon ;
- Compound statements use curly braces { }
- 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 in code

 1

Java Identifiers

- Required Identifier Naming Rules
  - Identifiers are case sensitive
  - Begin each identifier with a letter, _underscore, or $ dollar sign (no numbers)
  - May use numbers for any character after first
  - No spaces allowed, but may use underscore _
  - Identifier maximum length 255 is characters, but try to use less to minimize input errors
  - Not a Java keyword (approximately 50)
  - public class *name* must be same as *filename*.java
- Suggested Identifier Naming Guidelines
  - *Class identifier* should be TitleCase without spaces
  - *Method identifier* should start with lower case and be a verbNoun combination with words in title case
  - *Variable identifiers* are nouns and begin with lower case letter(s) to indicate variable data type

 2

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		

 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:

```
// The File Name for this program is Anatomy.java
public class Anatomy // Class definition header
{   // Class definition body begins here
  public static void main(String[] args)
  {   // main method is starting point of program
    System.out.println("Anatomy of a class");
  }   // main method code ends here
}   // Class definition body 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

 4

Output using PrintStream Class

- Package java.io
  - Multiple classes stored in same directory or folder
  - Class PrintStream contains methods
    - `println()  // Std output and ends with new line`
    - `print()  // Std output and no new line`
- Usage syntax:
  - `System.out.print("Enter Dimes Quantity: ");`
- *Escape Characters* can be contained in string
  - `\"` Double quote.
  - `\'` Single quote.
  - `\\` Backslash.
  - `\n` New line. Go to the beginning of the next line.
  - `\r` Carriage return. Go to beginning of current line.
  - `\t` Tab. White space up to the next tab stop.

 5

Escape Character Example

- How do you print characters with special meaning? For example, how do you print the following string?

  `The word "hard"`

  Would this work?

  `System.out.println( "The word "hard"" );`

  No, it would give a compiler error - it sees the string `The word` between the first set of double quotes and is confused by what comes after
- Use the backslash character, `\"`, to escape the special meaning of the internal double quotes:

  `System.out.println( "The word \"hard\"" );`

 6

Data Value Literals

- Literals are fixed human-readable values that can not be altered by program
  - Numbers
    - Integer Values are Whole Numbers

      1 -406 352563 0 -32 123456789
    - Floating Point Values are Real Numbers

      5. 0.0 -0.015 -1.5e-2 157.675 1.57675e2
  - Character Codes
    - Single Characters

      'A' 'a' 'C' '3' '$' '\n' '/' '?'
    - Strings of Characters

      "ABC" "abc\ndef" "32" "-5.2" "-1.5e-2"

 7

Java Arithmetic Operators

- *Arithmetic Operators*
  - *Precedence Order* is the order the operation
  - Parenthesis ( ) have highest precedence
  - Use parenthesis if order of operation not apparent

    (Precedence Highest to Lowest)

( )	Defines order of operation	→
-	Negative (unary)	←
* / %	Multiply, Division, Modulus	→
+ -	Addition, Subtraction	→
=	Assignment	

- Concatenation Operator +
  - For joining "Strings" and 'Characters'
  - "Hot " + "Dog" + '\n' + "That\'s mine\n"

 8

```java
public class OperEx01
{
  public static void main(String args[])
  {
    System.out.println(100.0000);
    System.out.println(6);
    System.out.println(3.75);
    System.out.println(100+25);
    System.out.println(-100+25);
    System.out.println(100-25);
    System.out.println(100*25);
    System.out.println(-100/25);
    System.out.println(-100/-25);
    System.out.println(100/31);
    System.out.println(100%31);
    System.out.println(100.0/31.0);
    System.out.println(1e2%3.1e1);
    System.out.println(6.5/2.1);
    System.out.println(6.5%2.1);
  }
}
```

```
> java OperEx01
100.0
6
3.75
125
-75
75
2500
-4
4
3
7
3.225806451612903
7.0
3.095238095238095
0.19999999999999973
```

Mixed Mode Expressions

- *Integer Expression*
  - If <u>all</u> numbers are integers then result is integer
- *Real (Floating Point) Expression*
  - If <u>any</u> number is floating point (real) then result is floating point (real) number
- *String Expression*
  - If any value on either side of the + operator is a string then the operator is concatenation
  - You can force an arithmetic operation by enclosing the Integer or Real Expressions with Parenthesis
  - *ASCII*: 8-bit, Latin characters (Limited to C++ but Not Java)
    - Both uppercase and lowercase letters
    - Digits 0 to 9 and keyboard symbols $,#.!;@*
  - *Unicode*: 16-bit, All Language Glyphs, Java!
    - 65,536 different glyphs for all languages

 10

Compound Operations

```java
public class OperEx02
{
  public static void main(String args[])
  {
    System.out.println(3+5+7);
    System.out.println(5*6+3);
    System.out.println(3+5*6);
    System.out.println(5*(6+3));
    System.out.println(-6*7%3+2);
    System.out.println(-6*7%(3+2));
    System.out.println(6*4+3*2);
    System.out.println(6*(4+3)*2);
    System.out.println(6*(4+3*2));
    System.out.println(100/8*2);
    System.out.println(100%8/3);
  }
}
```

```
> java OperEx02
15
33
33
45
2
-2
30
84
60
24
1
```

 11

```java
public class OperEx03
{
  public static void main(String args[])
  {
    System.out.println(20/3);
    System.out.println(20./3);
    System.out.println(3.+9/6);
    System.out.println(3+9/6.);
    System.out.println("ABC"+'D'+"EF");
    System.out.println("ABC"+'\t'+"EF");
    System.out.println("ABC"+'\"'+"EF");
    System.out.println("Product = " + 7*5);
    System.out.println("Quotient = " + 7/5.);
    System.out.println("Remainder = " + (7%5));
    System.out.println("Sum = " + 7+5);
    System.out.println("Sum = " + (7+5));
    System.out.println("Difference = " + (7-5));
    System.out.println("23 + 42 = " + 23+42);
    System.out.println("23 + 42 = " + (23+42));
  }
}
```

```
6
6.666666666667
4.0
4.5
ABCDEF
ABC    EF
ABC"EF
Product = 35
Quotient = 1.4
Remainder = 2
Sum = 75
Sum = 12
Difference = 2
23 + 42 = 2342
23 + 42 = 65
```

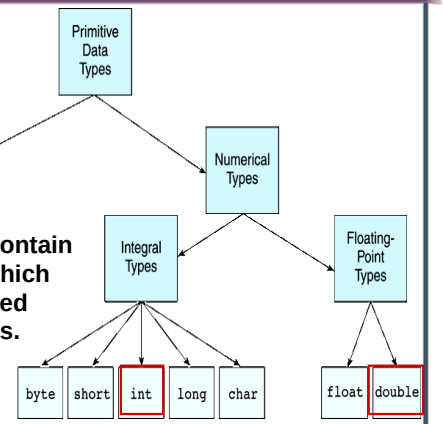
Java Console Application with Variables

```
/* Invoice Application
 * Author: Robert Laurie
 * Date: 14 March 2012
 * ------------------------------------
 * Description: This program will
 * calculate the invoice price using
 * a 20% discount
 */
public class InvoiceApp
{
   public static void main(String[] args)
   {
      // VARIABLE DECLARATIONS
      double dDiscount, dInvoice, dOrder = 2000;

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

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

```
C:\javacode>javac InvoiceApp.java

C:\javacode>java InvoiceApp

INVOICE APPLICATION
Order total:   $2000.0
Discount:      $400.0
Invoice total: $1600.0

C:\javacode>_
```

 13

Java – Primitive Data Types



Primitive data types contain *atomic data values*, which can not be decomposed into smaller data types.

Most common DataTypes are int and double

 14

Declaration Statements

- Variable is a container for data nLength 6
- *Declaration Statements* allocate memory to hold a data in a *Variable*
  - Specifies *Data Type*
  - Followed by *Identifier*

```
int nLength;
```

  - Terminate Java statement with semicolon ;
  - Optionally, may declare several variables of the same data type (comma separated)

```
int nLength, nArea;
```

  - Optionally, may initialize variables in declaration statement

```
int nLength = 6;
```

 15

Integer Data Type Declarations

- int
  Reserves 32 bits (4 bytes) of RAM memory which can represent:
  - Range of values ≈ ± 2 billion
    2,147,483,647 to -2,147,483,648
  - Declaration Examples:

int nSSN = 390546348;

nSSN 390546348

int nTotalScore, nClassMedian;

nTotalScore nClassMedian

int nAltitude = -100, nDistance = 50000;

nAltitude -100 nDistance 50000

 16

Long, Short, and Byte Integer Data Types

- **long**
  Reserves 64 bits (8 bytes) of RAM memory
  - Range of values ≈ ± 9 quadrillion
    9,223,372,036,854,775,807 to -9,223,374,036,854,775,808
  - `long lnDistance = -350L;`

  lnDistance **-350**
- **short**
  Reserves 16 bits (2 bytes) of RAM memory
  - Range of values: +32,767 to -32,768
  - `short snTotal=400, snScore=1;`

  snTotal **400** snScore **1**
- **byte**
  Reserves 8 bits (1 byte) of RAM memory
  - Range of values: +127 to -128
  - `byte bnPercent = 12;`

  bnPercent **12**

 17

Floating Point (Real) Data Types

- **float**
  Reserves 32 bits (4 bytes) of RAM memory
  - Range of values ≈ ± 1 x $10^{\pm 38}$ (7-digit precision)
  - ± 3.4028234 x 10^{+38} to ± 1.4012984 x 10^{-45}
  - `float fCash = 257.5F;`
- **double**
  Reserves 64 bits (8 bytes) of RAM memory
  - Range of values ≈ ± 1 x $10^{\pm 308}$ (15-digit precision)
  - ± 1.7697693134862315 x 10^{+308} to
    ± 4.940656458412465 x 10^{-324}
  - `double dCash = 257.5, dSavings = 2.5e6;`

 18

Precision, Exponential Notation, Atomic Data

- **Precision:**
  - Significant digits of a number
  - Significant digits determine Accuracy
  - Fewer digits results in round-off error
- **Exponential notation:**
  - Scientific Notation format
  - 63421.0 can be written 6.34210e4
  - 0.00634210 can be written 6.34210e-3
- **Atomic data versus Composite data**
  - Atomic data is complete entity by itself
  - Composite data is composed of multiple data entities that are atomic or composite for example a string of characters

 19

Number DataType Example

```
1.  public class DataTypeEx01
2.  {
3.    public static void main(String args[])
4.    {
5.      int nNum1 =300,  nNum2 = 1000;
6.      double dNum4 = 7.0, dNum5 = 10, dNum6;
7.      float fNum7 = 7f, fNum8 = 10F, fNum9;
8.      System.out.println(nNum1 + " " + nNum2 );
9.      System.out.println(dNum4 + " " + dNum5);
10.     System.out.println(nNum2 / nNum1);
11.     System.out.println(dNum5 / dNum4);
12.     System.out.println(dNum5 / nNum1);
13.     System.out.println(nNum2 / dNum4);
14.     dNum6 = dNum5 / dNum4;
15.     System.out.println(dNum6);
16.     fNum9 = fNum8 / fNum7;
17.     System.out.println(fNum9);
18.     System.out.println("Done");
19.   }
20. }
```

```
300 1000
7.0 10.0
3
1.4285714285714286
0.0333333333333333
142.85714285714286

1.4285714285714286

1.4285715
Done
```

 20

Character and Boolean Data Type

- **char** Reserves 16 bits (2 bytes) of RAM memory
  - Unsigned integer in range 0 to 65535 representing character
  - Examples:

```
char cGradeA = \u0065, cGradeB = 'B';
```

- **boolean** Reserves 1 bit of RAM memory
  - Usually, 1 byte because smallest addressable memory size
- Evaluates as **true**/**false**

```
1.  public class DataTypeEx02
2.  {
3.    public static void main(String args[])
4.    {
5.      char cGradeA = 65, cGradeB = 'B', cGradeC = '\u0043';
6.      boolean bRaining = true;
7.      System.out.println(cGradeA + " " +cGradeB+" " + cGradeC);
8.      System.out.println("Is it Raining? " + bRaining);
9.    }
10. }
```

```
A B C
Is it Raining? true
```


Primitive Data Types (Size and Range)

Type Name	Identifer Prefix	Literal Postfix	Kind of Data	Memory Allocated	Data Range
byte	bnVar		integer	1 byte	-128 to 127
short	snVar		integer	2 bytes	-32768 to 32767
int	nVar	default	integer	4 bytes	-2,147,483,648 to 2,147,483,647
long	lnVar	123L	integer	8 bytes	-9,223,372,036,854,775,808 to 9,223,374,036,854,775,808
float	fVar	12.5f 12.5F	floating point	4 bytes	+/- 3.4028… x 10^{+38} to +/- 1.4023… x 10^{-45}
double	dVar	default	floating point	8 bytes	+/- 1.767… x 10^{+308} to +/- 4.940… x 10^{-324}
char	cVar	'A'	Single character (Unicode)	2 bytes	65,536 Unicode characters
boolean	bVar		true or false	1 bit	not applicable

DataType Literals

- Literals are fixed human-readable values that can not be altered by program

LITERALS	DATA TYPE
'A'	Char
"Hello"	String of characters
+3 12 -123	Integer (no literal)
35000L -35L	Long Integer
123.45F -4.1e-2f	Float
123.45 -4.1e-2	Double (no literal)
0x4F 0x6B 0x21	Hexadecimal (Base 16)
026 001	Octal (Base 8)

Constants and Type Cast Operators

- Named Constants are immutable; can't change
  - final double TAXRATE_MI = 5.5;
  - final int TOTAL_STATE = 50;
  - final double PI = 3.141592653589793;
  - final float PI = 3.1415927;
- *Type Casting* operator changes the data type
  - Syntax: (dataType) expression
  - dHalf = (double) 1 / 2;
  - nDimes = (int) dTotal * 10;
  - dChipValue = (double) nChips * 20;
  - bnCreditCards = (byte) (nAMEX + nVisa + nMC);
  - lnTotalDebt = (long) nDebt;
  - nDebt = (int) lnTotalDebt;

