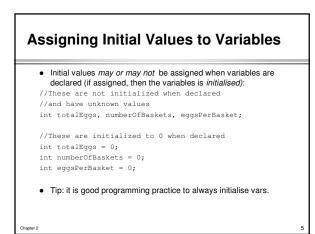


- The assignment operator: "=" (the "equals" sign)
 » not the same as in algebra (that is the "==" operator)
- It means: "assign the value of the expression on the RHS to the variable on the LHS." (RHS-right hand side)
- Variable can be on both sides of the equals sign, but only the left side obtains the assignment: int count = 10;// initialize counter to ten count = count - 1;// decrement counter
 value of count at the end: 10 - 1 = 9





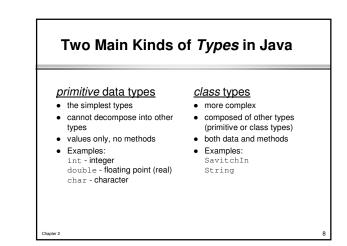
- value of variable is changed (assigned a different value) somewhere in the program
- may be calculated from other values: totalEggs = numberOfBaskets * eggsPerBasket;
- or read from keyboard input: totalEggs = SavitchIn.readLineInt();

Variable Names: Identifiers

Rules

- these must be obeyed • all Java identifiers follow the
- same rules (not only vars.)
- must not start with a digit must contain only numbers,
- letters, underscore (_) and \$ (avoid using \$, it is reserved for special purposes)
- names are case-sensitive (ThisName and thisName refer to different identifiers)

- Good Programming Practice - these should be obeyed always use meaningful names from the problem domain (for example, eggsPerBasket instead of n, which is meaningless, or count,
- which is not meaningful enough) • start variable names in lower case
- (classes, objects with uppercase)
- capitalise interior words (use eggsPerBasket instead of eggsperbasket)
- avoid using \$ since it is reserved for special purposes



Type Name	Kind of Value	Memory Used	Size Range
byte	integer	1 byte	-128 to 127
short	integer	2 bytes	-32768 to 32767
int	integer	4 bytes	-2,147,483,648 to 2,147,483,6
long	integer	8 bytes	-9,223,372,036,854,775,808 t 9,223,374,036,854,775,808
float	floating point	4 bytes	+/- 3.4028 x 10 ⁺³⁸ to +/- 1.4023 x 0 ⁻⁴⁵
double	floating point	8 bytes	+/- 1.767 x 10 ⁺³⁰⁸ to +/- 4.940 x 0 ⁻³²⁴
char	single character (Unicode)	2 bytes	all Unicode characters
boolean	true or false	1 bit	not applicable

Which Ones to Know for Now

Display in text is for reference; for now stick to these simple primitive types:

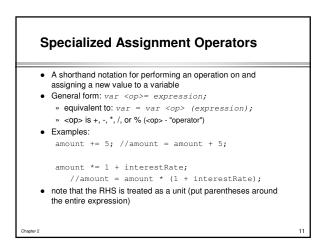
double

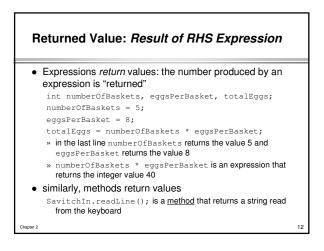
- int
 - » just whole numbers
 - » may be positive or negative
 - » no decimal point
- char
 - » just a single character

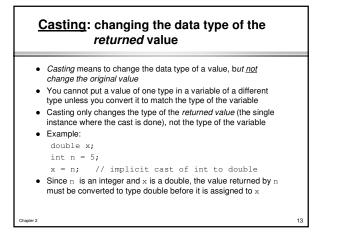
 - » uses <u>single</u> quotes
 - » for example char letterGrade =
 - `A`;
- positive and negative » has a decimal point (fractional part) » two formats

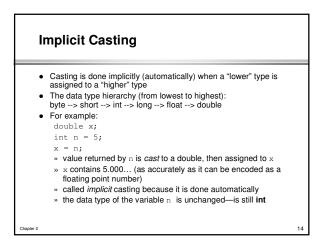
» real numbers, both

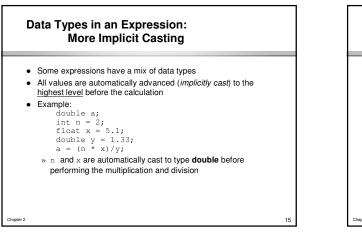
- number with decimal point, e.g. 514.061 – e (or scientific, or
- floating-point) notation, e.g. 5.14061 e2, which means
- 5.14061 x 10²

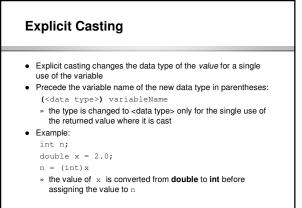


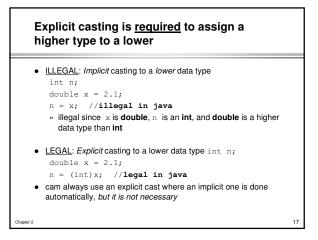


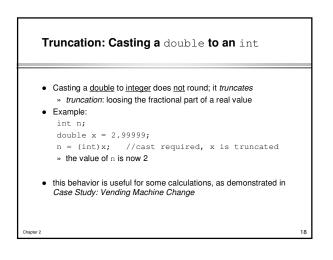


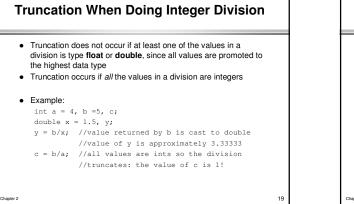


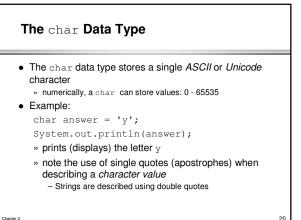












Characters as Integers Characters are actually stored as integers according to a special code each printable character (letter, number, punctuation mark, space, and tab) is assigned a different integer code the codes are different for upper and lower case for example 97 may be the integer value for 'a' and 65 for 'A' ASCII (Appendix 3) and Unicode are common character codes Unicode includes all the ASCII codes plus additional ones for languages with an alphabet other than English

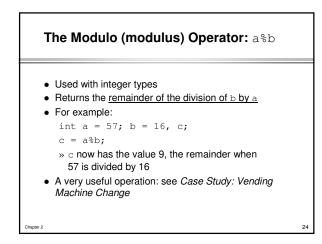
• Java uses the Unicode character set

Casting a char to an int • Casting a char value to int produces the ASCII/Unicode value • For example, what would the following display? char answer = 'y'; System.out.println(answer); System.out.println((int)answer);

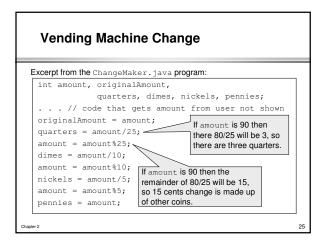
- Answer: the letter 'y' on one line followed by the ASCII code for 'y' (lower case) on the next line:
 - 89

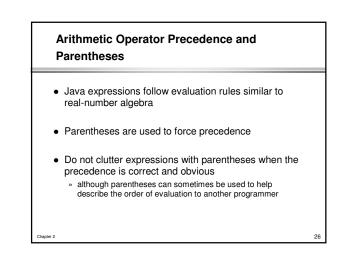
GOTCHA: Imprecision of Floating Point Numbers

- Computers store numbers in variables that are defined by a fixed number of bits, so real (floating point, fractional) values can not be encoded precisely,
 - » an infinite number of bits would be required to precisely represent any real number
- But since Integers have no fractional values, the are encoded precisely,
- » if 2 is assigned to an int variable, the value is precisely 2
 Knowing which data type to use where is important in programming (as will been seen later in the course)



22

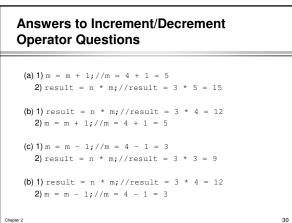


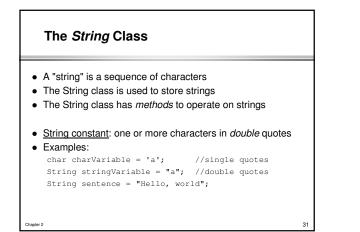


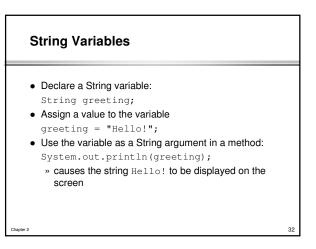
Examples of Expressions						
Ordinary Math Expression	Java Expression (preferred form)	Java Fully Parenthesized Expression				
rate ² + delta	rate*rate + delta	(rate*rate) + delta				
2(salary + bonus)	2 * (salary + bonus)	2 * (salary + bonus)				
time + 3mass	1/(time + 3 * mass)	1/(time + (3 * mass))				
$\frac{a-7}{t+9v}$	(a - 7) / (t + 9 * v)	(a - 7) / (t +(9 * v))				
	·					
Chapter 2			27			

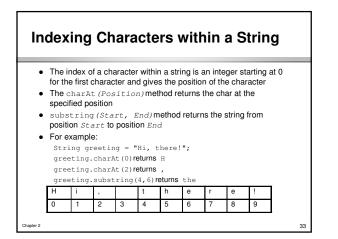
» used printCounters m	notation for common arithmetic operations on variables marily for <i>counting</i> ay "count" up(+) or down(-), but <u>only with integer variables</u>
	an be incremented (or decremented) before or after using value; example,
int cour	
++count	: preincrement count: count = count + 1 before using it
count++	: postincrement count: count = count + 1 after using it
count	: predecrement count: count = count -1 before using it
	: postdecrement count: count = count -1 after using it

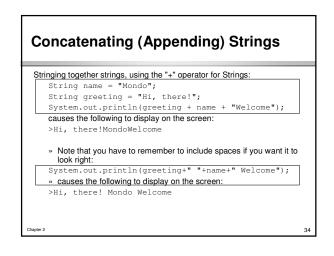
Increment and Decrement Operator Examples	Ans Ope
<pre>code segment, int n = 3; int m = 4;</pre>	(a) 1) 2)
int result; What are the values of m and result after each statement?	(b) 1 2)
<pre>(a) result = n * ++m; //preincrement m (b) result = n * m++; //postincrement m (c) result = n *m; //predecrement m</pre>	(c) 1 2)
<pre>(d) result = n * m; //postdecrement m</pre>	(b) 1) 2)
*2 29	Chapter 2

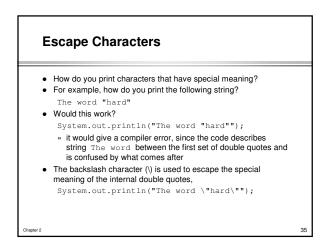


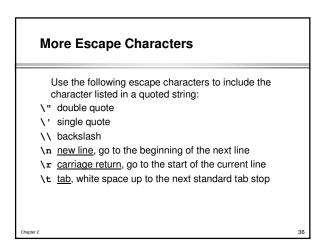


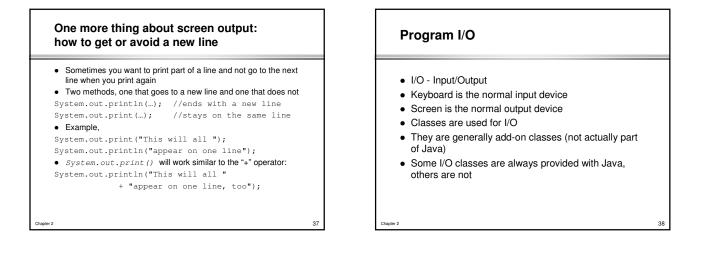








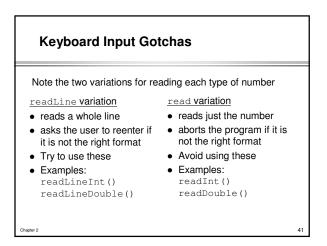


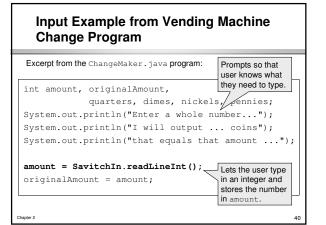


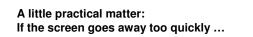


- We have been using an output method from a class that automatically comes with Java: System.out.println()
- But Java does not automatically have an input class, so one must be added
 - » SavitchIn is a class specially written to do keyboard input
- SavitchIn.java is provided with the text see Appendix 2
 Examples of SavitchIn methods for keyboard input:
- readLineInt() readLineDouble()
- readLineNonwhiteChar()
- Gotcha: remember Java is case sensitive, for example readLineNonWhiteChar() will not work

Chapter :







If the output (screen display) of your programs does not stay on the screen, use this code,

System.out.println("Press any key to end program.")
String junk;

junk = SavitchIn.readLine();

- The display stops until the user presses ENTER
- Whatever the user types is stored in the string ${\tt junk}$ but is never used
- Note: As seen in lab using JCreator (or another Java IDE), at the termination of a program, the IDE pauses the screen anyway

Documentation and Style

- Use meaningful names for variables, classes, etc.
- Use indentation and line spacing as shown in the examples in the text
- · Always include a "prologue" (an brief explanation of the program at the beginning of the file)
- Use all lower case for variables, except capitalize internal words (eggsPerBasket)
- · Use all upper case for variables that have a constant value, PI for the value of pi (3.14159...) (see text for more examples)

Named Constants

- Named constant-using a name instead of a value • Example: use MORTGAGE_INTEREST_RATE instead
- of 8.5
- · Advantages of using named constants,
 - » easier to understand program, reader can tell how the value is being used
 - » easier to modify program, value can be changed in one place (the definition) instead of being changed everywhere in the program.
 - » avoids mistakes of changing a similar value that is used for a different purpose

Defining Named Constants public static final double PI = 3.14159; public-no restrictions on where name can be used static-required (explained later) final-program is not allowed to change the value • the remainder of the definition is similar to a variable declaration and gives the type, name, and initial value. • such a declaration is provided at the beginning of the program file and is not inside the main method definition



- · Comments-program text ignored by the compiler
- · Does not change what the program does, but used to explain the functionality and steps in a program
- Always include meaningful and useful comments
- Goal is to comment the non-obvious statements
- Assume a reasonably knowledgeable reader
- // single-line comments (everything after is ignored)
- /* ... */ multi-line comments

Summary

- Variables hold values and have a type
- » data type of a Java variable is either a primitive or a class
- » common primitive types include int, double, and char
- » [so far] a common class type is String
- » before being used, variables must declared
- Parentheses in arithmetic expressions ensure correct evaluation
- Input/Output
 - » Output: use System.out.print() /println() methods
 - » use SavitchIn methods for keyboard input
 - SavitchIn is not part of standard Java

Summary Good programming practice:

- » Use meaningful names for variables
 - » Always initialize variables

 - » Use variable names (in upper case) for constants
 - » Use comments sparingly but wisely, e.g. to explain nonobvious code
 - » Prepare the program so that it looks visually attactive indenting, line spacing, appropriate use of comments
 - » Display prompts should precede any user input statements (so that the user knows what to enter)