* Understanding java operators:
  + Java operator is a special symbol that can be applied to a set of variables, values or literals.
  + Three types of operators are available in Java: unary, binary, and ternary
  + If 2 operators have the same precedence, then java guarantees elft-to-right evaluation
  + Order of operator precedence:
    - Post- unary operators: expression++, expression—
    - Pre-unary operators: ++expression, --expression
    - -,+,~,!
    - \*./,%
    - +,-
    - <<,??,???
    - <.>.<=,>=,instance of
    - ==,!=
    - &,^,|
    - &&,||
    - Boolean expression ? expression1: expres-sion2
    - Assignment operators
* Working with Binary Arithmetic Operators
  + Most common operators in the java language
* Arithmetic operators: +,-,\*,/,%++,--
  + Parenthesis always takes precedence
  + PEMDMAS (parenthesis, exponents, multiplication, division, modulus, addition, subtraction)
* Numeric promotion
  + Numeric promotion rules
    - If two values have different data types, Java will automatically promote one of the values to the larger of the two data types
    - If one of the values s integral and the other is floating-point, java will automatically promote the integral value to the floating point values data type
    - Smaller data types, namely byte, short, and char are first promoted to int any time they are used with a java binary arithmetic operator, even if neither of the operands is int.
    - After all promotion has occurred and the operands have the same data type, the resulting value will have the same data type as its promoted operands
  + Floating point literally are assumed to be a double unless postfixed with an f, as in 2.1f
* Working with unary operators:
  + A unary operator is one that requires exactly one operand, or variable, to function.
  + They usually increase/decrease numeric values by one or negate Boolean values
  + Examples of unary operators: +,-,++,--,!
  + You cannot apply a negation operator – to a Boolean expression
  + You can’t apply ! to a numeric expression
* Using additional binary operators:
  + Int x = 1;
* Casting primitive types
  + Int x = (int)1.0;
  + Short y = (short)1921222; //stored as 20678
  + Int z = (int)9f;
  + Long t = 192301398193810323L;
* Compound assignment operators:
  + +=,-=
  + Compound operators can save us from having to explicitly cast a value.
  + Long x = 10, int y = 5; y\*=x; y will be casted to a long and apply the multiplication of two long values and then cast the result to an int
* Relational operators:
  + <,<=,>,>=
  + Relational instance of operator: a instanceof b – true if the reference that a points to is an instance of a class, subclass, or class that implements a particular interface, as named in b
* Logical operators
  + &,|,^ can be applied to both numeric and Boolean data types
  + When they are applied to Boolean data types, they are referred to as logical operators.
  + When they are applied to numeric data types, they are referred to as bitwise operators, as they preform bitwise comparison of the bits that compose the number.
  + X & Y (And): true only if both operands are true
  + X | Y (Inclusive or): false if both operands are false
  + X ^ Y (Exclusive or): true if the operands are different
  + && short-circuit operator
  + || short-circuit operator
  + &&,|| are identical to logical operators except that right-hand side of the expression may never be evaluated if the final result can be determined by the left hand side of the expression.
    - Example: x = true || (y<4);
  + A more common example of where short-circuit operators are used is checking for null objects before performing an operation, such as this
    - If (x!=null && x.getValue()<5){//do something}
      * If x was null, then the short-circuit prevents a null pointer exception from ever being thrown, since evaluation of x.getValue() is never reached
      * However, if && was replaced with & then it would throw a null pointer exception
* Equality operators:
  + There is a difference between two objects are the same and two objects are equivalent
  + == or != operators are used in 1 of three scenarios:
    - Comparing two numeric primitive types. If the numeric values are of different data types, the values are automatically promoted as previously described
      * 5 == 5.00 returns true since the left side is promoted to a double
    - Comparing two Boolean values
    - Comparing two objects, including null and string values.
    - For object comparison, the equality operator is applied to the references to the objects, not the objects they point to. Two reference are equal, if and only if they point to the same object or both point to null.
* Understanding Java statements (Java control flow statements):
  + Control flow statements break up the flow of execution by using decision making, looping, and branching, allowing the application to selectively execute particular segments of code.
* The if-ten-else statement:
  + Order is important when creating if-then-else statements.
* Ternary operators: