U N I V E R S I T Y

CS 180: Problem Solving and Object-Oriented Programming

Lecture 3: Numbers, Operators, Types, and Conversions

Dr. Jeff Turkstra

Announcements

- Office Hours
  - Tuesday/Thursday 10:00am – 11:30am
  - ...and by appointment
  - Happy to meet virtually, just send me an email
- Homework 1 posted
- Remember to attend your assigned lab session this week!

Lecture 03

- Values, variables, and literals
- Types
- Assignment
- Statements, expressions, blocks
- Conversion and casting
- Characters

UseCalculator.java

```
import java.util.Scanner;

public class UseCalculator {
  public static void main(String[] args) {
    Calculator c = new Calculator();

    Scanner scanner = new Scanner(System.in);
    int x = scanner.nextInt();
    int y = scanner.nextInt();
    System.out.println(c.add(x, y));
  }
}
```


Formatting notes

- Naming conventions
  - Variables: lowerCamelCase
  - Classes: UpperCamelCase
  - Symbolic constants: UPPER_CASE
- Open curly at end of line
- Consistent indentation (eg, 2 spaces)
- Complete documentation on course website

Assignments

- “Create a class Henway...”
  - Create a file Henway.java that contains the class declaration

```
public class Henway {
}
```

  - Note capitalization
  - Compile it
  - Run it

Good Day, World!

```
import javax.swing.*;

class HelloWorld {
  public static void main(String[] args) {
    JFrame myWindow;
    myWindow = new JFrame();

    myWindow.setSize(300, 200);

    myWindow.setTitle("Good Day, World!");
    myWindow.setVisible(true);
  }
}
```


- "At the source of every error which is blamed on the computer you will find at least two human errors, including the error of blaming it on the computer"

Values, variables, literals

- Programs and CPUs work with values and addresses
- Values are represented in programs by literals
- Addresses are represented by variables
- Values are stored in variables

Literals

- Source code representation of a fixed value
  - No computation required

```
3, -23, 4.5, 0.23, 3E8, 6.02e+23
0xbeefbeef, 0b11010
1234_5678_9012_3456L, 5_2
```

  - Not `3_.1415F, 999_L, 55_`

```
1.234f, 123.4d, 2.34D
"Hello there"
'A'
true, false
```


Variables

- x, y, a, b, helloMessage, wheel, robot, r1, w27
- Composed of letters, digits, and _
  - Start with a letter
- Identify a memory location
  - Instance variables
  - Class variables
  - Local variables
  - Parameters

Types

- Tell the compiler or interpreter about a piece of data
- Include its representation and a set of operators for manipulating the representations
- Variables and literals both have types

Java and types

- Java is a statically typed language
  - Every variable and expression has a type known at compile time
- Java is strongly typed
  - Types limit the values that a variable can hold, the operations supported, and their meaning
- Two types: primitive types and reference types

Primitive types

- Predefined and named by a reserved keyword
- boolean
- Integral types
  - byte (8-bit)
  - short (16-bit)
  - int (32-bit)
  - long (64-bit)
  - char (16-bit unsigned integer, UTF-16)
- Floating-point types
  - float (32-bit IEEE 754)
  - double (64-bit IEEE 754)

Integer operations

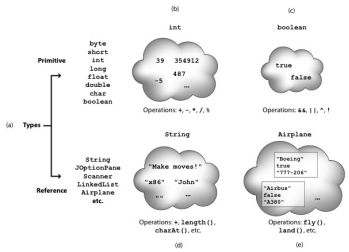
- Comparisons
  - <, <=, >, >=, ==, !=
- Unary plus and minus (+ and -)
- Multiplicative *, /, and %
- Additive + and -
- Increment ++ (prefix and postfix)
- Decrement -- (prefix and postfix)
- Signed and unsigned shift <<, >>, >>>
- Bitwise complement ~
- Bitwise operators &, ^, and |
- Conditional operator ?
- Cast operator
- String concatenation operator +

Floating-point operations

- Pretty much the same
  - No bitwise fun

boolean type

- Result of any comparison
- Operators are:
  - == and !=
  - Logical complement !
  - Logical &, ^, and |
  - Conditional-and/or && and ||
  - Conditional operator ?
  - String concatenation +
    - "true" or "false"
- Only expression usable for control flow
  - if, while, do, for, conditional ?



Reference types

- Four kinds:
  - Class
  - Interface
  - Type variables
  - Array types

class Object

- Superclass of all other classes
- All class and array types inherit its methods
  - clone
  - equals
  - finalize
  - getClass
  - hashCode
  - wait, notify, notifyAll
  - toString

class String

- Sequences of Unicode code points
- String literals are references to instances of class String
- Concatenation (+) implicitly creates a new String object
  - If the result is not a compile-time constant
- Methods include
  - concat() or +
  - toUpperCase()
  - length()
  - substring()
  - etc

Remember Wheel?

```
public class Wheel {
    double radius;

    Wheel(double radius) {
        this.radius = radius;
    }

    double getCircumference() {
        return 2 * Math.PI * radius;
    }

    double getArea() {
        return Math.PI * radius * radius;
    }

    double getRadius() {
        return radius;
    }
}
```


Wheely

- Set of values
  - Limited by memory only
  - New one created for each new Wheel(...)
  - Same or different radius
- Set of operations
  - getArea()
  - getCircumference()
  - getRadius()

Assignment

- Note that = is not the same as mathematical equality
- It is an assignment

```
circumference = 2 * 3.14 * radius;
```

Can declare and initialize at the same time

```
int a = 7, b = 5;
```

Expressions

- Combine variables, literals, operators and method invocations
- Constructed according to language syntax
- Evaluates to a single value

Precedence

- Subexpression ( )
- Postfix (expr++, expr--)
- Unary (++expr, --expr, +expr, -expr ~ !)
- Multiplicative *, /, %
- Additive +, -
- Shift <<, >>, >>>>
- Relational <, >, <=, >=, instanceof
- Equality ==, !=
- Bitwise AND &
- XOR ^
- OR |
- Logical AND &&
- Logical OR ||
- Ternary ? :
- Assignment =, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=

Statements

- Analogous to sentences
- Complete unit of execution
- Often expressions terminated with a semicolon
  - Expression statements
- Declaration statements
- Control flow statements

Blocks

- Group of zero or more statements between balanced braces
- Can be used anywhere a single statement is allowed

Purdue trivia

- "The first dean of agriculture, John Skinner, did some of Purdue's most effective lobbying in the Indiana General Assembly, armed with Purdue enthusiasm and a bushel or two of ripe apples from a university orchard."

  - A Century and Beyond, by Robert W. Topping

Conversion

- Every expression has a type
  - Based on literals, variables, and methods
- Conversions can happen at compile time and run time
  - int to long
- Only some conversions are permitted

Conversions

- Identity
- Widening primitive
- Narrowing primitive
- Widening reference
- Narrowing reference
- Boxing
- Unboxing
- Unchecked
- Captures
- String
- Value

Widening

- byte to short, int, long, float, or double
- short to int, long, float, or double
- char to int, long, float, or double
- int to long, float, or double
- long to float or double
- float to double

Wider

- Sometimes called type promotion
- Usually does not lose information
  - float to double can lose magnitude information
  - int or long to float may lose precision
- Never results in a runtime exception

Narrowing

- short to byte or char
- char to byte or short
- int to byte, short, or char
- long to byte, short, char, or int
- float to byte, short, char, int, or long
- double to byte, short, char, int, long, or float

Narrower

- May lose information about overall magnitude as well as precision and range
  - double to float
  - Signed integer to integral type (byte, short, int, long, char)
- Often not permitted without a cast

Casting

- Converts, at runtime, a value of one numeric type to a similar value of another numeric type
  - Except boolean
- Checks, at runtime, that a reference value refers to an object of compatible class
- You are telling the compiler: “I know better than you do what should happen here.”

- Upcasting – type promotion, widening
  - Permitted and can happen automatically
- Downcasting – narrowing
  - Dangerous
  - Precision is lost
  - Prevented by default
  - Must be done explicitly

Autoboxing

- Primitive types have corresponding "object wrapper" classes
  int → Integer, double → Double, etc

```
Character myChar = 'a';
```

- Conversion the other way is called unboxing

char

- Remember, everything is a number (0s and 1s)
- 'A' = 65, 'B' = 66, 'a' = 97, etc
- 16-bits
- Building block of Strings

- ASCII subset, 0-127
  - English alphabet, numbers, punctuation, special characters
- Latin-1 extension, 128-255
  - non-English (Romanized) characters and punctuation (dipthongs, accents, circumflexes, etc)
- Unicode 1 extension, 256-65535
  - non-Romanized alphabetic characters from Cyrillic, Hebrew, Chinese, Japanese, etc

char Literals

- Single character surrounded by single quotes
  - 'A', 'a', 'x', '0', '!', ',', '"', '&'
- Escape sequences
  - '\t' tab
  - '\n' newline
  - '\'' single quote
  - '\\' backslash
  - '\uxxxx' hexadecimal xxxx from unicode

Operations

- Autoboxes to Character
  `System.out.println('A');`
  "Hello" + '!' → "Hello!"
- Remember .toString()?
- Treated as integer for arithmetic operations
  - 'a' + 0 → 97
  - 'z' - 'a' → 25

Character methods

isDigit(char)
isLetter(char)
isLetterOrDigit(char)
isLowerCase(char)
isUpperCase(char)
isWhiteSpace(char)
toLowerCase(char)
toUpperCase(char)

Boiler Up!