COBOL: COBOL Concept Description
Java: Java/OO Similar Concept
++: What Java/OO adds to Concept

COBOL: Load Module/Program
Java: Class

COBOL: PERFORM
Java: method
++: can pass parameters to method, more like FUNCTION
other programs/classes can call methods in different classes if declared public. public/private gives designer much control over what other classes can see inside a class.

COBOL: Working Storage, statically linked sub-routine

Java: instance variables

++: (see next)

COBOL: Working Storge, dynamically loaded sub-routine

Java: Class variables

++: Java can mix both Class variables (called static, just the reverse of our COBOL example, and instance variables (the default).

COBOL: PICTURE

Java: No real equivalent.

I therefore invented a method to mymic a ZZZ,ZZZ,... mask for integer input. Here is an example of padLeft that implements this logic. padLeft is also a good example of polymorphism. In COBOL, if you have different parameter lists you need different entry points. In Java, the types of parameters are part of the definition. For example:

```
 * paddedString  = u.padLeft(oldString,10);   // pad left with blanks
 * paddedString  = u.padLeft(oldInt,10);      // comma inserted every 3 bytes.
 * paddedString  = u.padLeft(oldInt,10,2);    //    " + .00 (2 is # decimal points).
 * paddedString  = u.padLeft(oldLong,10);     // comma inserted every 3 bytes.
 * paddedString  = u.padLeft(oldLong,10,2);   //    " + .00 (2 is # decimal points).
```

All the padLeft methods do essentially the same function. However, the ones that accept int or long values, will also insert the comma in every 3 bytes, and suppress leading zeroes. The number of decimal digits is my way of handling the issue of decimal rounding. This will work as long as you add or subtract integers with the same assumed decimal precision, and, if you multiply or divide, you manually handle the scaling.

COBOL: Decimal arithmetic
Java: Not in native Java, but IBM has implemented some BigDecimal classes.

COBOL: COPY or INCLUDE
Java: Inheritance
++: Much more powerfull!

COBOL: ON EXCEPTION
Java: try/throw/catch
++: can limit scope of error detection (see following)

COBOL: OPEN
Java: Input Streams
++: Automatic error detection, both a blessing and a curse.

COBOL: WRITE
Java: write (yes, really).

COBOL: CLOSE
Java: close method

COBOL: READ
Java: read...