**Variables in interfaces**

**import** java.util.Random;

**interface** SharedConstants{

**int** *NO*=0;

**int** *YES*=1;

**int** *MAYBE*=2;

**int** *LATER*=3;

**int** *SOON*=4;

**int** *NEVER*=5;

}

**class** Question **implements** SharedConstants{

Random rand = **new** Random();

**int** ask(){

**int** prob = (**int**) (100 \* rand.nextDouble());

**if** (prob < 30)

**return** *NO*; // 30%

**else** **if** (prob < 60)

**return** *YES*; // 30%

**else** **if** (prob < 75)

**return** *LATER*; // 15%

**else** **if** (prob < 98)

**return** *SOON*; // 13%

**else**

**return** *NEVER*; // 2%

}

}

**public** **class** AskMe **implements** SharedConstants{

**static** **void** answer(**int** result){

**switch**(result){

**case** *NO*:

System.*out*.println("No");

**break**;

**case** *YES*:

System.*out*.println("Yes");

**break**;

**case** *MAYBE*:

System.*out*.println("Maybe");

**break**;

**case** *LATER*:

System.*out*.println("Later");

**break**;

**case** *SOON*:

System.*out*.println("Soon");

**break**;

**case** *NEVER*:

System.*out*.println("Never");

**break**;

}

}

**public** **static** **void** main(String args[]){

Question q = **new** Question();

*answer*(q.ask());

*answer*(q.ask());

*answer*(q.ask());

*answer*(q.ask());

}

}

**Result:**

Yes

Yes

Yes

Soon

(The result is not absolute)

You can use interfaces to import shared constants into multiple classes by simply declaring

an interface that contains variables that are initialized to the desired values. When you

include that interface in a class (that is, when you “implement” the interface), all of those

variable names will be in scope as constants. (This is similar to using a header file in C/C++

to create a large number of **#defined** constants or **const** declarations.) If an interface

contains no methods, then any class that includes such an interface doesn’t actually

implement anything. It is as if that class were importing the constant fields into the class

name space as **final** variables. The example above uses this technique to implement an

automated “decision maker”.

The method nextDouble() is used. It returns random numbers in the range of 0.0 to 1.0.

**Interfaces can be extended**

**package** practice;

**interface** A{

**void** meth1();

**void** meth2();

}

**interface** B **extends** A{

**void** meth3();

}

**class** Methods **implements** B {

**public** **void** meth1(){

System.*out*.println("Meth 1 performed");

}

**public** **void** meth2(){

System.*out*.println("Meth 2 performed");

}

**public** **void** meth3(){

System.*out*.println("Meth 3 performed");

}

}

**class** MyClass{

**public** **static** **void** main(String[] args){

Methods methods = **new** Methods();

methods.meth1();

methods.meth2();

methods.meth3();

}

}

As an experiment, you might want to try removing the implementation for **meth1( )** in

**MyClass**. This will cause a compile-time error. As stated earlier, any class that implements

an interface must implement all methods required by that interface, including any that are

inherited from other interfaces.

One remarkable thing I accidently learned from this is, the class’ name and the main class’ names must match.