**Data Types– Part 1**

**Slide 1**

In part one of this presentation on data types we will discuss scalar data types. Variables of scalar data types contain only a single value.

**Slide 2**

The first kind of scalar type that we will consider are those for numeric types. Numeric types can be categorized into integer and real types.

First we consider integer types—those that contain whole numbers.

Most languages provide several sizes of integer types, typically in integral bytes. Java has four sizes that include byte, short, int and long. They are one, two, four and eight bytes respectively. C and C++ allow the last three and also treat the type char as an integral type, but the actual sizes are machine dependent.

C and C++ allow both signed and unsigned integers. Type coercion is allowed between the two kinds of types which can result in some unexpected results because neither kind is a subset of the other.

Ada’s approach is quite different. Although Ada has predefined integer types of more than one size, it also allows for the definition of subtypes and new types with specific ranges. In addition, it allows modular integer types on which modular arithmetic can be performed.

This is an example of a new integer type for the days in a month with a specific range. Because it is declared as a new type it is not type compatible with the predefined Integer type. It is also possible to constrain the range of an integer type while maintaining compatibility with the predefined Integer type by using a subtype instead. In either case, providing range constraints allows the compiler’s run-time system to generate a run-time error should a variable of this type be assigned a value outside the range.

Every programming language provides some data type for real numbers. Most languages provide only a floating point representation. Most provide more than one level of precision.

Ada is somewhat unique in that it allows both floating point and fixed point types.

In addition to providing predefined floating point types, Ada allows user-defined types. This declaration specifies both a precision and a range. The digits clause specifies the precision by requiring 10 significant digits.

In fixed point type declarations, an accuracy must be provided. This delta clause indicates that the type Money is accurate to one cent. A range can also be provided although one was not included in this example.

**Slide 3**

Next we consider predefined nonnumeric types.

Characters are the first kind of nonnumeric type. In some languages, like those in the C family, the character type is type compatible with the integer type. By contrast, in Ada, they are not.

Character literals in most languages are written by enclosing them in single quotes. Nonprintable characters require a special representation. In the C family escape characters are used.

In Ada, the Character type is a predefined enumerated type. The fact that character literals can be enumeration literals makes that possible. A nonprinting character like a carriage return is represented by the name ASCII.CR.

Some languages have a distinct Boolean type for logical values.

In C and C++, bool is not a distinct type, but a typedef of int.

0 represents false.

Everything else represents true.

In Ada, the Boolean type is a predefined enumeration type.

**Slide 4**

Finally we consider enumerated types.

Enumerated types are user-defined nonnumeric types.

Pascal, Ada and C++ all had enumerated types as a part of their original language.

By contrast, Java did not include them initially but added them in version 5. In Java their implementation includes some features that some other languages did not.

All enumerated types inherently extend the predefined class Enum.

Consequently they inherit all methods defined in that class, which includes a method that returns the string that names a literal and another that returns the ordinal value of an enumeration literal.

Like classes, other user-defined methods can be defined.

In addition, abstract enumerated types can be defined that are akin to a hierarchy of singleton objects.

**Slide 5**

Let’s look at an example of an enumerated type in Java that includes a user-defined method.

This method returns a Boolean value indicating whether a color is considered a warm color, which would include red, orange and yellow. It makes use of the method ordinal that is inherited from the Enum class.

Because the Enum class implements the Comparable interface, so does every enumerated type. So we can call the compareTo method on enumeration literals. Their ordering is defined by the order in which they are listed in the type definition.

The next statement is a call to the method isWarm that we included in the enumeration definition above.

The call to ordinal will return an integer that represents the position of the literal in the type definition.

Finally the name method returns a string that corresponds to the name of the enumeration literal.

Here is the output we would see if we ran the program. It outputs the fact that red is less than blue, blue is not a warm color, the ordinal value of red is 0, and it displays the string *BLUE* for the color blue.

**Slide 6**

Finally, we will examine an enumerated type declaration for colors implemented in Ada.

The type definition itself consists of only the enumeration literals. Enumerated types belong to a category of types called discrete types, which automatically provide a collection of attributes. Included among those attributes are certain constants, operators and functions.

All six relational operators are defined for all discrete types and are therefore available with this type. The ordering is the order in which they are listed in the declaration.

The next two functions are also implicitly defined for all discrete types. The function Pos is equivalent of Java’s ordinal method. Note the use of the single quote here. This syntax is used whenever an attribute associated with a type category is selected.

Lastly, the function Image is the equivalent of Java’s name method.