

## **Ph265. Compiled languages; First Java Program; Introduction to type-cast and Object-Oriented programming.**

### ***Interpreter and Compiled languages***

When working with Maple, we enter commands "line by line" and have the computer execute the commands in the same manner. This way of execution is convenient when we need to quickly check some assumption, plot a graph of a known function, or do some other tasks which do not require extensive time-consuming computations. In a sense Maple acts like an "interpreter" – trying to check our spelling and execute our commands one by one. This immediately means that if we managed to make a mistake somewhere at the end of long list of computations – we will have to wait for a computer to get to the last line and discover the mistake.

Another approach to scientific computing is to use so-called "compiled" languages. In this case we first use a text editor to write our program. We then call a compiler – which checks the program for "spelling" and "grammar" (like ":", ";") errors and converts our text file into a binary code – directly executable by a computer with no further conversion. Since all spelling mistakes in this approach are corrected at first step, the execution is takes much shorter time. Moreover – the mistakes may be discovered even before we first run the program. On the other hand – the additional "compilation" step requires time – thus the compiled languages are not the best for "short" and "quick" calculations.

### ***Class – the building block of an object-oriented language***

In computing the data is usually stored in some variables. In Maple, and in many other languages the variables, once defined, are accessible throughout the program. Any function, defined in the text can modify any data. This may potentially lead to modification of inappropriate data at inappropriate time. Moreover, since the type of the data in Maple is defined at the moment we use the variable (and even can be changed during the program), it is possible to "accidentally" modify the data in the wrong way. Java, C++, and some other languages use a different strategy. First – we have to define the type of each variable before using it. Second – the type of each variable is fixed throughout the program and compiler checks whether some specific operation is compatible with some specific data type (for example, we may think twice before adding two "character" variables). Third – these languages allow us to combine data and methods of operating these data in one single structure – class. A class can be anything – thermometer (which would contain single variable – temperature, and methods on reading this variable), speedometer, control valve, timer, particle counter, etc. Class ideology prevents the user from application of wrong methods on the wrong data *at the compilation time*.

Next – when we define a class we must create an *object* of this class to use it. Generally, each object has its own copy of the data. As an example, a house can have large number of thermostats – one for each room. Similarly, the scientific project can operate a number of sets of data. Thus, the accelerator experiment can have one set of data per run. Each

such set will be processed independently of other sets of data, but the same algorithm will be used for the processing.

Another useful concept is the one of derived classes. Given one class we can create several of its "children" – or derived classes. Conceptually, a parent class must provide some general structure of all its children and realizations. Example – we again use thermostat. The parent class defines two data fields – current temperature and desired temperature, and four methods – turn A/C on and off, turn heater on and off. The "child" class may describe gas heaters and electric A/C (which would have additional functions, like turn pilot light on and off), electric heaters and A/C (with additional functions giving electrical power used). Note: each derived class may add data to its parent, and may change its functions – but can not remove "parent" functionality. Therefore – interface software can freely operate in terms of "parent" terminology – and the product will yield correct results, picking up correct methods *at the run time*.

## **Functions**

Functions are the base elements of operations on class data. Each function can introduce its own temporary variables to perform its algorithm. These variables may be summation or loop indexes, temporary sum values, etc. Functions may change the values of class data, and/or may return some values.

## **First Java program**

```
1 //Area.java: Area of a circle , sample program
2 public class Area {
3     public static void main(String args []){
4         //Instantiate variables
5         double radius , circum , area , PI = 3.14159265359;
6         int modelN = 1;
7         radius = 1.;
8         circum = 2.* PI* radius;
9         area = radius * radius * PI;
10        //Output the variables to the screen
11        System.out.println("Program number = " + modelN);
12        System.out.println("Radius = " + radius);
13        System.out.println("Circumference = " + circum);
14        System.out.println("Radius = " + radius +", Area = " + area);
15    }
16 }
```

This program has to be typed into a file called Area.java, then we have to open command line, and compile the program – calling "javac Area.java". This produces byte code "Area.class". The class can be run by "java Area".