

Introduction to Java

COMP1100 — Introduction to Programming and Algorithms

Reading: *Big Java Ch.2*

Clem Baker-Finch

Australian National University

Semester 1, 2006

Where We Are Going?

We have looked at some key ideas about programming, using [Haskell](#).

These key ideas also occur in [Java](#).

The rest of the course will be structured around these correspondences.

The idea is to emphasise the **concepts** by seeing them from another angle.

Important Correspondences:

types: types, classes

values: values, objects

functions, type signatures: functions, type declarations

lists: various list library classes

data-directed program design: data-directed program design

modules: classes

polymorphic types: generic classes

recursion: recursion, loops

map, fold, **etc:** for-each loops

show **function:** toString() method

read **function:** java.util.Scanner class

algebraic data types: abstract classes, inheritance

libraries: libraries

Defining Functions in Java

Remember this Haskell function definition?

```
restRate :: Double -> Double -> Double -> Double
restRate weight height age =
    (66.47+13.75*weight+5*height-6.76*age)*4.2
```

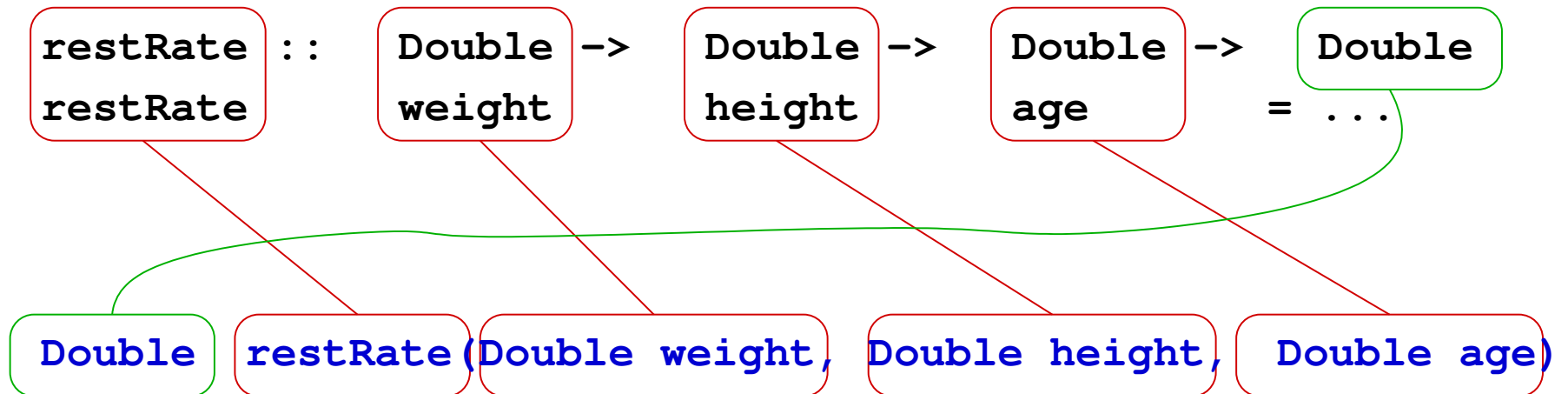
How do we write it in Java?

```
Double restRate(Double weight, Double height, Double age)
{
    return (66.47+13.75*weight+5*height-6.76*age)*4.2;
}
```

(Java has a Float type, but usually stops you using it...)

Type Signatures Correspond

The type signature and the left side of the defining equation in Haskell contain exactly the same information as the function definition header in Java:



(In Java, we always write the type before the identifier we are describing.)

Java Commands Describe Actions

In Haskell, the body of a function is just an **expression** which is evaluated to give the result value:

```
(66.47+13.75*weight+5*height-6.76*age)*4.2
```

In Java, the body of a function is a (sequence of) **commands** (statements).

So we have to *instruct the function to return a value* — it is an *action*.

Hence the **return** statement:

```
return (66.47+13.75*weight+5*height-6.76*age)*4.2;
```

Java is not Layout Sensitive

Some languages (like Haskell) use **indentation** to indicate where things like function definitions end.

Other languages (like Java) use **punctuation** to indicate where things like function definitions end.

The end of a statement (command) is indicated by a **semicolon**:

```
return ... ;
```

Things (e.g. statements) are grouped together using *braces*:

```
Double restRate (...) {  
    ...  
}
```

People Reading Java Programs are Layout Sensitive!

How readable is this?

```
public class Point{Double x; Double y; public Point()  
{x = 0.0;y =0.0;}public Point(Double xVal, Double  
yVal){x = xVal;y = yVal;}public Double xCoord  
(){return x;} public Double yCoord(){return y;}  
public void moveRight(Double distance){x=x+distance  
;}public void moveUp(Double distance){y=y +  
distance; }}
```

A Java Program?

Haskell *programs* all have a function:

```
main :: IO ()
```

which handled the input-output interactions.

Similarly, Java *programs* have a function:

```
public static void main(String args [])
```

which handles the input-output interactions.

Warning: If you thought Haskell I/O was difficult, wait until you see Java I/O!

[Example: RestRate1P.java]

Compiling Java Programs

The simplest way to compile a Java program is on the command line:

```
> javac RestRate1P.java
```

That is, `javac` is the Java compiler program.

A successful compilation produces a `.class` file with the same name (RestRate1P.class in this case).

Running Java Programs

Unlike `ghc` and many other compilers, `javac` does not produce a machine executable version of the program.

`Javac` produces a version of the program that runs on the *JVM* (Java Virtual Machine). The JVM is *another program*.

To run your (compiled) program on the JVM, use the command:

```
> java RestRate1P
```

(Make sure to leave off the `.class` suffix.)

Other ways to Experiment with Java Programs

Unfortunately, there isn't anything as convenient as GHCi for Java.

You might like to try *DrJava*, which does allow some experimentation with parts of programs.

Eclipse is a fully featured integrated development environment available on the student system. The COMP1110/1510 lecturer uses *Eclipse* quite a lot. (Eclipse also supports Haskell. . .)