

# Algebraic Data Types in Java

**COMP1100 — Introduction to Programming and Algorithms**

Clem Baker-Finch

Australian National University

Semester 1, 2006

## Algebraic Types in Haskell

Recall that one of the ways we can define our own types in Haskell is as *algebraic data types*.

The simplest kind of algebraic types are **enumerations**. For example:

```
data Day = Sun | Mon | Tue | Wed | Thu | Sat
```

```
data Temp = Cold | Hot
```

```
data Season = Spring | Summer | Autumn | Winter
```

Can we do that in Java?

## Enumerations in Java

```
enum Day {SUN , MON , TUE , WED , THU , FRI , SAT};
```

```
enum Temp {HOT , COLD};
```

```
enum Season {SPRING , SUMMER , AUTUMN , WINTER};
```

(It is a convention to write constants in upper case. It is not a rule.)

Java `enum` actually defines a new class, so you have to write `Season.SPRING`.

```
Temp weather(Season s) {  
    if (s.equals(Season.SUMMER))  
        return Temp.HOT;  
    else  
        return Temp.COLD;  
}
```

## Union Types

Haskell's algebraic data types can be more interesting than simple enumerations — the alternatives can be **structured**.

Here is a type of different shapes, together with their *dimensions*:

```
data Shape = Circle Float
           | Rectangle Float Float
           | Ellipse Float Float
           | Square Float
```

For example, (`Rectangle 12.0 8.0`) represents a rectangle of length 12.0 and height 8.0

In Haskell we define functions on union types, case by case:

```
area :: Shape -> Float
area (Circle rad)      = pi * rad^2
area (Rectangle l b)   = l * b
area (Ellipse maj min) = pi * maj * min
area (Square side)     = side^2
```

## Can we do that in Java?

Yes, but of course it looks a bit different.

### Pick out just one alternative first — Rectangle

Based on the Haskell code on previous slides, rectangles alone can be represented as:

```
data Rectangle = Rectangle Float Float
```

and we can define functions like this:

```
area :: Rectangle -> Float
area (Rectangle l b)    = l * b
```

(See `Rectangle.hs`)

## How to Represent Rectangles *Alone* in Java?

One of the first Java classes we looked at was `java.awt.Rectangle`.

Here we will build our own version:

```
class Rectangle {  
  
    Double length, height;  
  
    public Rectangle(Double l, Double h) {  
        length = l;  
        height = h;  
    }  
  
    ...  
}
```

Notice that in Haskell we refer to `Rectangle` (in the expression `Rectangle 12.0 8.0`) a ***constructor function*** as it *constructs a rectangle*.

Which corresponds exactly to the ***constructor*** (method) in the Java `rectangle` class. The expression `new Rectangle(12.0,8.0)` constructs the same rectangle.

Corresponding to the functions we define in Haskell, we have methods like

```
public Double area() {  
    return length * height;  
}
```

and so on.

## Lesson:

**We can build a Java class corresponding to each of the alternatives in a Haskell algebraic type.**

(Circle.java, Rectangle.java, Square.java, Ellipse.java)

In Haskell, that's like

```
data Circle      = Circle Float
data Rectangle   = Rectangle Float Float
data Ellipse     = Ellipse Float Float
data Square      = Square Float
```

which is not quite what we want.

In Haskell we can have variables of type `Shape` which may be circles, rectangles, ellipses *or* squares.

**How can we do that in Java?**

## Abstract Classes

**Abstract classes** don't have **objects**.

Abstract classes can be “**extended**” by concrete classes of the same “signature.” That is, the concrete classes implement all the methods of the abstract class.

(See `Shape.java`)

The methods in `Shape.java` are declared `abstract` and do't have bodies.

Each of the concrete classes `Circle.java`, `Rectangle.java`, `Square.java` and `Ellipse.java` starts like this:

```
class Rectangle extends Shape {
```

to indicate that it is related to `Shape`.

We can have variables, parameters and function results of type `Shape`.

For example:

```
public Shape normalise () {
```

which will return an object of class `Circle`, `Rectangle`, `Square` or `Ellipse`.

So we have achieved something corresponding to the Haskell algebraic data type.

A more complex example of an algebraic data type was used to represent pictures in the *ANUPlot* library. Corresponding Java classes are given on the *Lectures* page of the course website.

## UML Class Diagrams

An aspect of **UML** (the Universal Modelling Language) is **class diagrams** to graphically represent the relationship between classes in a system. Classes are represented by boxes and the relations are represented by lines or arrows.

One of the main arrows represents the **“is-a”** relationship:



In the shapes example, each of **Circle**, **Rectangle**, **Square** and **Ellipse** **“is-a”** **Shape**.



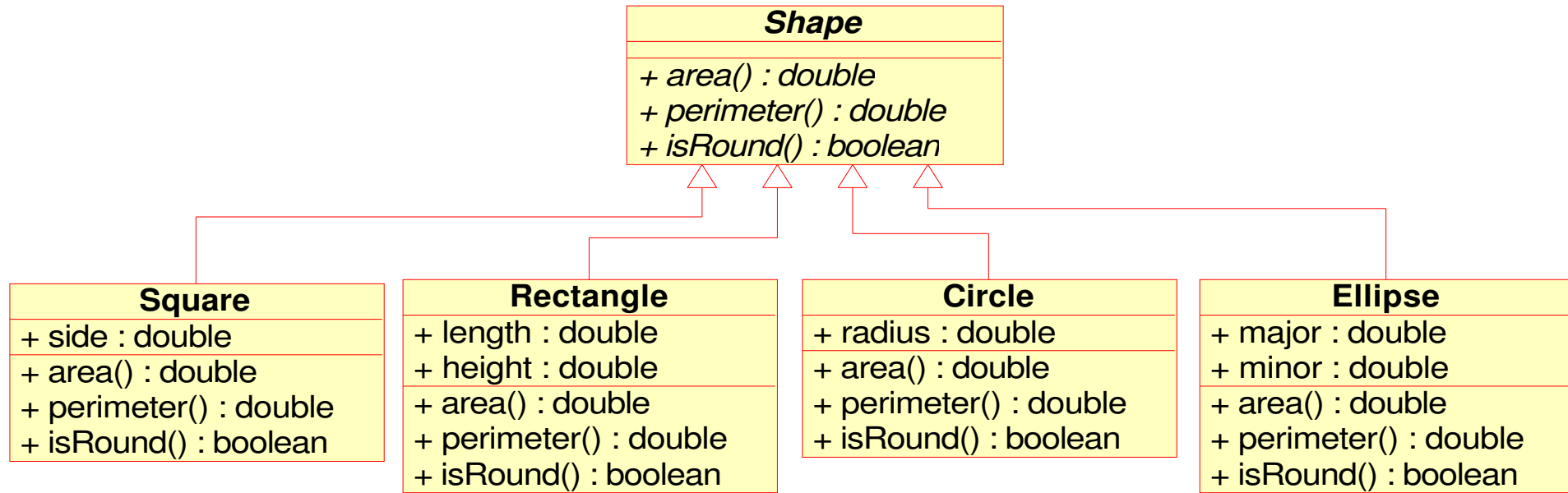


Diagram: class diagram Page 1

## The End.

- **Summary lecture Thursday 25 May 9.30am**
- **Sample exam questions in the next week or so...**
- **Watch course website for drop-in labs schedule between now and exam.**
- **Pre-exam massed tutorials/lectures in week prior to exam. Watch website for announcement.**
- Thanks and good luck.