SWINBURNE UNIVERSITY OF TECHNOLOGY

Introduction to Programming

Week 5, Topic 1: Complex Data Types

(Records, Enumerations)

KNOWING

Complex Data Types

- This lecture we start by looking at programmer created Data Types:
  1. Records/Classes
  2. Enumerations

Complex Data Types vs Primitive Data Types

- We have been using '**primitive**' data types which are **basic** types that come with the language Ruby (and most other languages).
- These primitive/basic datatypes are the essential ones you need to create more complex data types.
- What are the primitive data types we have been using so far?

Programmer Created Data Types

- Also known as '**custom**' data types.
- These are created to represent the entities you are modelling in your program.
- Each program is a model, or abstraction, of some aspects of the real world.
- As an abstraction, some details are left out, others are specifically included for the purpose of the model.

Complex Data Type Example

- Consider a student.
- What attributes does a student have?
- What attributes would be included in a student administration system for a university?

Example Student Record (Data Dictionary)

Field	Data Type	Complex/Basic	Example/format
id	String	Basic	1023450X
first_name	String	Basic	Jenny
last_name	String	Basic	Fortesecue
email_address	String	Basic	jforte@gmail.com
home_address	Address	Complex	Street number, street name, suburb, state, postcode, country.
DOB	Date	Complex	dd/mm/yycc
phone_number	String	Basic	0414 899 456
enrolled_course	String	Basic	BA-ICT
unit_results	Results	Complex	unit_code, mark, grade, semester, year (multiple)
unit_enrolments	Enrolment	Complex	unit_code, semester, year

Date complex data type

- Some complex data types are not ‘custom’ in the sense that the application programmer needs to create them.
- Some complex data types are in libraries provided with or for the language.
- Eg: Date (which exists already for Ruby)
- If you were to write this yourself you might start with a record that looks as follows:

Field	Type	Example
Day	Integer	31
Month	Integer	12
Year	Integer	2001

This is an improvement on one long integer.

How to Represent Complex Data Types?

- In Ruby we use **Classes** to represent records such as the Student and Date records above.
- Eg:

```ruby
class Date
  attr_accessor :day, :month, :year
end
```

- In C we use **structs** which would look as follows:

```c
typedef struct Date {
    int day;
    int month;
    int year;
} Date;
```

Each of these has three attributes (day, month, year).

Structs in Ruby

- Ruby also has a form of struct:

```
Student = Struct.new(:name, :id, :email, :course)

s = Student.new("Sam", "0320", "sam@uni.com.au")

puts "Name: " + s.name
puts "Id: " + s.id
puts "Email: " + s.email
```

Type/Token Distinction

- A Class (or record or struct) represents a **type**.
- To allocate memory we need to create a **token**.
- We need to create an instance of the type.
- Eg for Ruby:

```
date = Date.new()
```

- And for C:

```
Date date;
```

Using complex data types.

- Once we create an instance (token) of a complex data type, we need to initialize its field/**attribute** values (which we should do with all variables):
- Eg for Ruby:

```
date = Date.new()
date.day = 31
date.month = 11
date.year = 2008
```

- And for C:

```
Date date;
date.day = 31;
date.month = 11;
date.year = 2008;
```

Initialising Complex Data Types

- In Ruby we can create a block of code to initialise the fields/attributes Eg for Ruby:

```ruby
class Date
  attr_accessor :day, :month, :year

  def initialize(day, month, year)
    @day = day
    @month = month
    @year = year
  end
end
```

- Use this as follows:

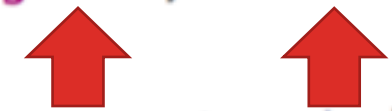
```ruby
date = Date.new(31, 11, 2008)
```

Nested Complex Data Types

- Below is a complex data type:

```
class Album
  attr_accessor :title, :artist, :genre, :tracks

  def initialize (title, artist, genre, tracks)
    @title = title
    @artist = artist
    @genre = genre
    @tracks = tracks
  end
end
```



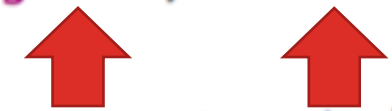
These are both custom data types.

Nested Complex Data Types

- Below is a complex data type:

```
class Album
  attr_accessor :title, :artist, :genre, :tracks

  def initialize (title, artist, genre, tracks)
    @title = title
    @artist = artist
    @genre = genre
    @tracks = tracks
  end
end
```



These are both custom data types.

Enumerations

- Enumerations are a custom data type that holds constant values.
- Enumerations simply assign meaningful names to a set of integers.
- Eg: to Represent different Genres of music we could use 0, 1, 2, 3 etc. Or we could create an enumeration as follows:

```
module Genre
  Pop, Classic, Jazz, Rock = *0..3
end
```

- Pop = 0, Classic = 1, Jazz = 2 and Rock = 3.

In C, this would be:

```
enum genre_names {Pop, Classic, Jazz, Rock};
```

Enumerations

- An enumeration you have already seen is one for the Z order of objects you draw on the screen in Gosu:

```
module ZOrder
  BACKGROUND, PLAYER, UI = *0..2
end
```

- This substitutes 0 for BACKGROUND, 1 for PLAYER and 2 for UI.

End of Topic 1.

Terminology:

- Primitive data types
- Complex Data Types
- Custom Data Types
- Enumerations
- Records
- Classes