Multi-Paradigm Programming - Object Oriented Programming

Dominic Carr

Galway-Mayo Institute of Technology

dominic.carr@gmit.ie

What We Will Cover

1. Goals of this Session

2. Object-Oriented
   - What is it?
   - Examples of OOP
   - Contrasted with Procedural
   - Message Passing
   - Impurity in OOP
   - Inheritance
   - Polymorphism
   - Interface
   - Composition
   - OOP Languages

Goals of this Session

Goals

- To understand....
  - What is Object Oriented porgramming?
  - How is it different?
  - How does OOP acheive abstraction, isolation, and reusability?
  - The basics of the Java programming language
  - Difference from Python
  - How to write an OOP Program in Java

Paradigms I

Recall

- The term **programming paradigm** is the style or way of **thinking about and approaching problems**.
- The chosen paradigm affects how the code is **written and structured**.
- It can heavily influence how **one thinks** about the problem being solved

Object-Oriented

Object-Oriented I

- Objects are representation entities e.g. Lists, Animals etc.
- An object has both state and functionality
- The class of an object can be thought of as it's template
  - Can specify what sort of states the object can have
  - What functionality the object can perform
- For example a Person is a class of Object, and "Dominic" is an instance of that class
  - State could include name, age, occupation, gender etc.
  - Each person will vary in these states

Object-Oriented II

- Object-Oriented programming builds up libraries of reusable Objects (code)
- Some OOP languages are "pure", eveything in the language is an object
  - Ruby is a pure OOP language
    - One of it's design goals was to have "everything as an object" and to be more OOP than python
    - Though since 2.2 python has increased it's purity `http://www.python.org/download/releases/2.2/descrintro/`
  - Java and C++ are impure OOP languages
  - As "primitive" data types are not objects in those languages

Object-Oriented III

Listing 1: Person Class in Java

```
public final class Person {
   private final String name;
   private final int age;

   public Person(String name, int age) {
       this.name = name;
       this.age = age;
   }
   public String toString() {
       return name + " " + info;
   }
   public static void main(String[] args) {
       Person a = new Person("Alice", 12);
       Person b = new Person("John", 34);
   }
}
```

Object-Oriented IV

Listing 2: Person Class in Ruby

```
class Person
        def initialize(name, age)
                @name = name
                @age = age
        end

        def name
                # labelname is only in scope within the name
                    method
        labelname = "Fname:"
        return labelname + @name
        end
end
```

Object-Oriented V

- Different languages, same paradigm
- We see some syntactic differences
- Ruby is dynamically typed
- Java is strongly typed
- Java is more verbose
- The first method shown in each is a special one
  - Referred to commonly as the constructor
  - Creates new instances of objects

Object-Oriented VI

Listing 3: C Example of representing a Person

```
#include <stdio.h>
struct person
{
   int age;
   float weight;
};
int main()
{
    struct person *personPtr, person1;
    personPtr = &person1;
    printf("Enter age: ");
    scanf("%d", &personPtr->age);
    printf("Enter weight: ");
    scanf("%f", &personPtr->weight);
    printf("Age: %d\n", personPtr->age);
    return 0;
}
```

Object-Oriented VII

- A struct is a grouping of variables under a single name
- It is a data-type itself, variables can be declared of it's type
- A struct has state, but not functionality
- Modifications to the state of a struct would be done by methods outside of the struct
- This is a major difference when the complexity of the programs grow.

Object-Oriented VIII

Message Passing

- Message passing is the means by which objects communicate with each other
- Most typically realised as method calls
  - written before runtime
  - but this is not always the case, some languages have more flexibility
- The format is typically:

```
receiver_object.method_name(argument1, argument2, ...)
```

- This can be understand as we are send a message to the *receiver_object* to perform *method_name* with our given input.

Object-Oriented IX

Message Passing

- For Example:

```
math_solver.add(12, 30)
```

- We shall discuss in future cases where the object need not explicitly define the exact message handler ahead of time.

Object-Oriented X

Java: Impure

- In Java Primitives are not objects.
  - This includes the data-types: int, char, float, and double.
- This was done for efficiency purposes.
- What is the consequence?
  - We cannot pass them messages, we cannot invoke a method on primitives

    ```
    1.toString()
    ```

- The above would not be valid Java, however in Ruby

  ```
  1.to_s()
  ```

- Is perfectly valid as "1" is an object.
- This is arguable nicer for the programmer as it brings consistency to the language

Object-Oriented XI

Java: Impure

- This impurity was "dealt with" somewhat with the introduction of object wrappers for the primitives
  - Such as "Double" which is an object version of "double".
  - When Java needs to treat them as objects "auto-boxing" is performed to convert them
- Such wrappers are a specail kind of object, they are **immutable objects**
  - When it has a value, the value cannot be changed.
  - Another example would be the **String** class in Java
- Immutability is sometimes a very useful property for certain objects

Object-Oriented XII

Inheritence

- Inheritance is a relation between two classes.
- Superclass is inherited from
- Subclass is the inheritor, it gets the properties of the superclass
- Inheritance promotes reusability

```
class Mammal
  def breathe
    puts "inhale and exhale"
  end
end

class Lion < Mammal
  def speak
    puts "Roar!!!"
  end
end
```

Object-Oriented XIII

- Many OOP languages will allow only single inheritence, where a sub-class inherits state and functionality from only one parent class
  - There are cases where multiple inheritence is supported through mixins
  - Also it is possible for an object to take on additional roles by implementing an interface or through “ducktyping”

Object-Oriented XIV

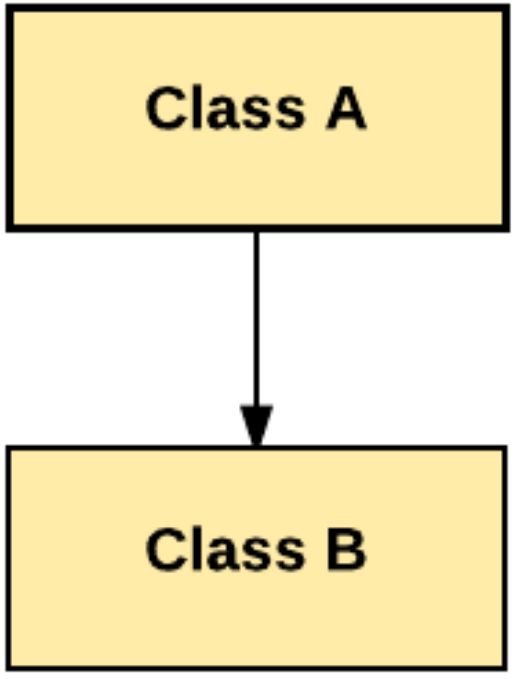

Figure: Single Inheritance

Object-Oriented XV

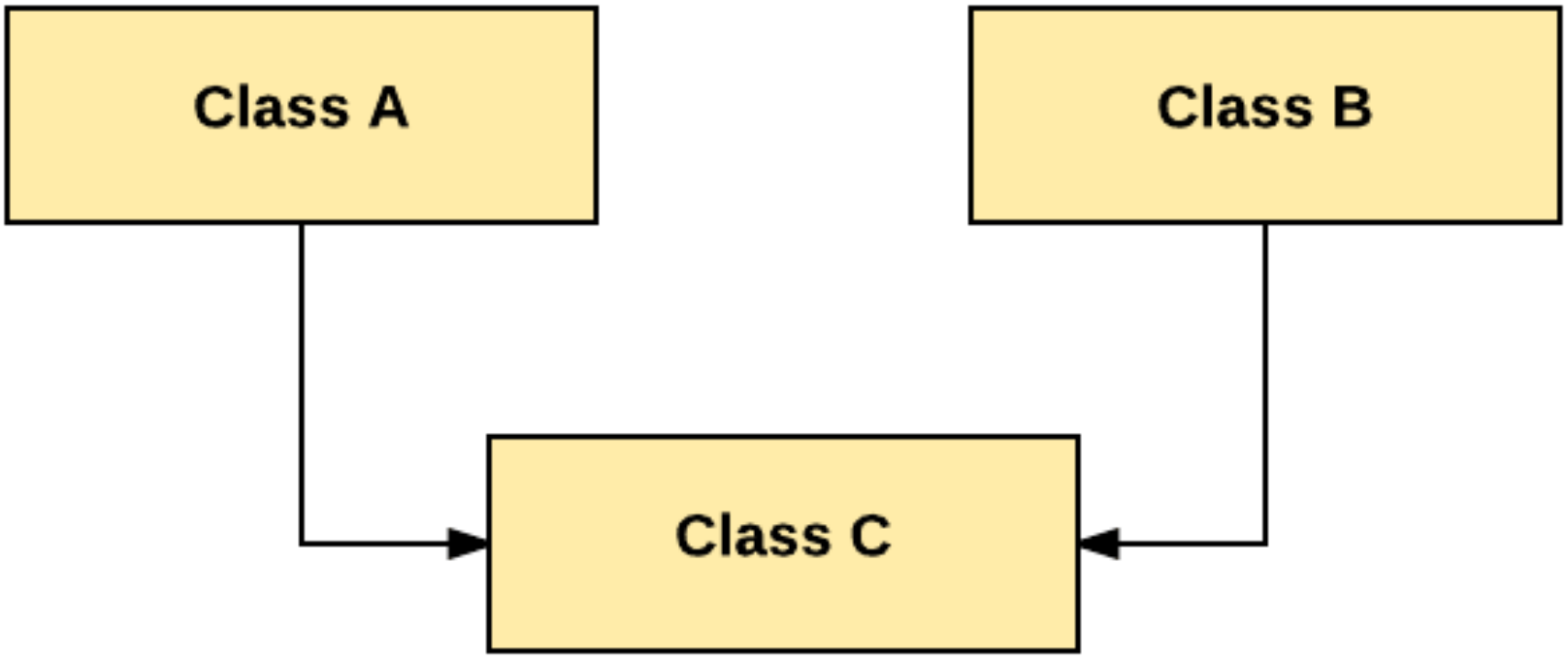

Figure: Multiple Inheritance

Object-Oriented XVI

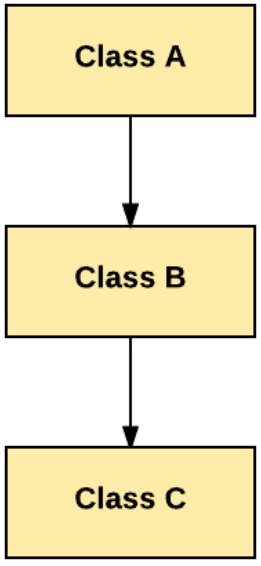

Figure: Multi-level Inheritance

- Class C inherits from B and it gets what B inherited from A
- In Ruby and Java (many others) all objects **implicitly** inherit from some base type.
  - In Java this is the class **Object**

Object-Oriented XVII

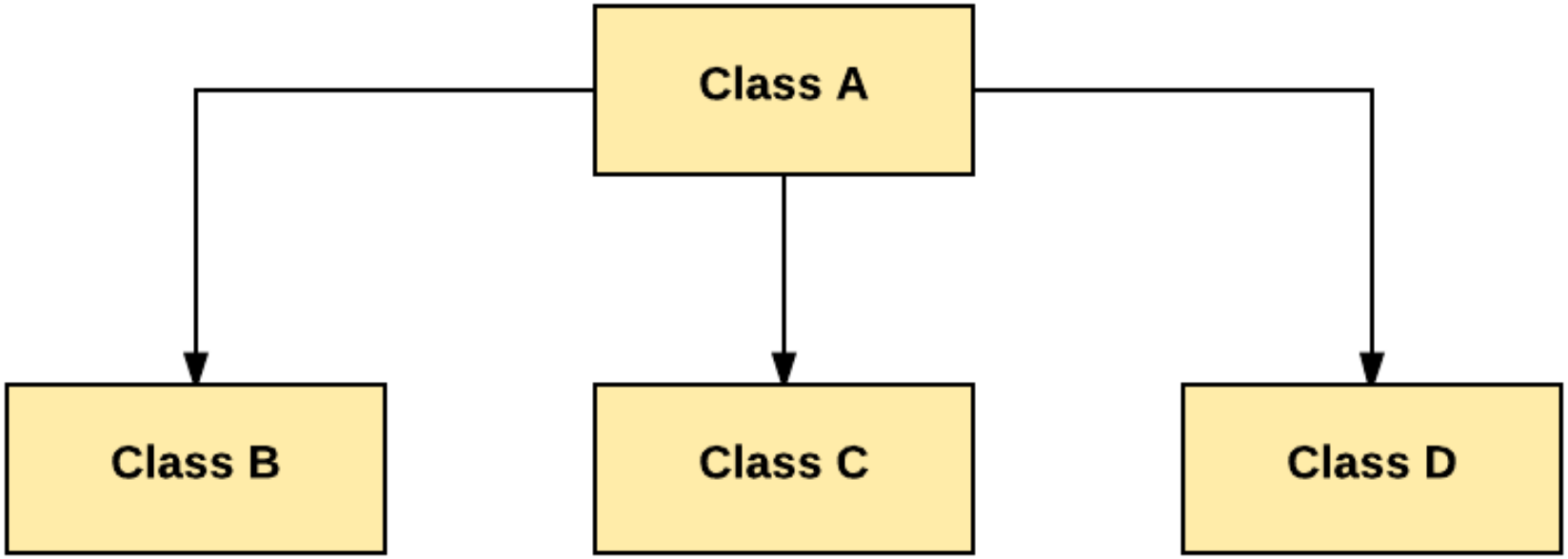

Figure: Hierarchical Inheritance

Object-Oriented XVIII

- The hierarchy is like what we had with the Mammal class

```
class Mammal
  def breathe
    puts "inhale and exhale"
  end
end

class Lion < Mammal
        .......
end

class Monkey < Mammal
        .......
end
```

- Both the Lion and the Monkey are Mammal and they both inherit from the Mammal class.

Object-Oriented XIX

- “Poly” means “many”
- Polymorphism is the ability of an object to take on many forms.
- The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
- For example a Monkey is a Mammal so when a reference is needed to a Mammal any Monkey can be substituted in.
- If a Lion was explicitly required then only it’s sub-types could be subsituted.
- It is a very important concept and we will discuss it further we we take a deeper look at the Java language

Object-Oriented XX

Interface

"an interface is a shared boundary across which two or more separate components of a computer system exchange information"

- Your mouse is an interface to the computer
- It moves the pointer around the screen
- But the same can be acheived with a track pad, or some kind of eye tracker etc.
- The point is the machine doesn't care what is moving the pointer as long as the correct signals are sent from the input device (mouse, trackpad) to control the machine (OS).
- Developers regularly use Application Programming Interfaces (API) to communicate with webservices such as Twitter, Facebook, TVMaze, Weather etc.

Object-Oriented XXI

- A class which implements an interface does not gain any state from the interface, it merely takes on the requirement to implement the "contract" of the interface. It must do (or delegate) that which the interface promises to it's users.
- It is possible for an object to inherit from a parent and implement a separate interface at the same time.
- We will see examples of this when we loom at Java and other languages.

Object-Oriented XXII

- An object can contain another object as an instance variable
  - Known as Object Composition
  - For example An Employee Object may have an Address object to represent their address
  - This stops the Employee class from being too complex as the Address class can deal with maintaining the address information and providing suitable methods for modifying.
- Object composition is used to represent “has-a” relationships
  - Every employee has an address
  - Every Student has many classes (So a Student object may have an Array of Module objects)

Object-Oriented XXIII

- OOP Languages
  - Python
  - Java
  - Ruby
  - PHP
  - C#
  - Objective C
  - Go
  - and many more!

Sources

- https://www.computerhope.com/jargon/i/imp-programming.htm
- https://en.wikipedia.org/wiki/Abstraction_(computer_science)

The End