# INFO322/IAPP001 Applications Programming

Tracy Quick Tracy.Quick@insearch.edu.au

## Polymorphism

## Learning Outcomes

- Students should be able to:
  - Define polymorphism, giving examples
  - Describe dynamic dispatch and its role in polymorphism

## Polymorphism

- What is Polymorphism?
- Dynamic Dispatch
- Non-polymorphism

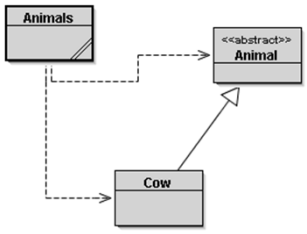
## What is Polymorphism ?

- Polymorphism means many forms. In OO, it means that a characteristic can have a different meaning or behaviour in a different context.
  - Different objects can react differently when the same method is called.

## What is Polymorphism ?

- Example: Consider a group of instruments
  - Each instrument can be played, but the method of play is different for each instrument.
  - A guitar is played in a different way to a piano
  - So, guitar objects behave differently to piano objects when the method play() is called because Guitar & Piano classes have a different implementation for the play() method

**Remember this list example from week 5 ?**




## Week 5 Revision

- Animals class has a list of Animal objects ie Cow objects.

  ***private LinkedList<Animal> animalList;***
- The Cow class inherits from Animal and <u>effects</u> (provide implementation for) noise() the abstract method in the parent class.
- Attribute animalList is of type LinkedList where the objects have been declared as Animal (so we do not need to cast)

  *private LinkedList<Animal> animalList;*

## Week 5 Revision

- The animalList attribute is created and populated in the Animals default constructor

```
animalList.add(new Cow("Clarabelle"));
animalList.add(new Cow());
animalList.add(new Cow("Buttercup"));
animalList.add(new Cow("Daisy"));
animalList.add(new Cow("Meg"));
```

- There are now 5 elements in the List

## Week 5 Revision

- Animals:noise() gets each element in the list and calls noise() on each element.

```
for(Animal current : animals)
{
   System.out.println(current.noise());
}
```

## Week 5 Revision

```
Clarabelle the Cow says moo
A Cow says moo
Buttercup the Cow says moo
Daisy the Cow says moo
Meg the Cow says moo
```

## Lists Example

- In Week 4, we looked at a list that held objects of the same type, Rectangle, Animal (only using Cow objects)
- But a Sheep is an Animal, so we can populate the animals with Sheep objects too!

## Example of a List

- The Sheep class inherits from Animal and effects (provide implementation for) noise() the abstract method in the parent class.
  - **Abstract method noise() in Animal class – line ??**
  - **Sheep inherits from Animal – line ??**
  - **Sheep default constructor – line ??**
  - **Sheep alternate constructor – line ??**
  - **Sheep effects noise() – line ??**

## Revisit Lists Example

- See Lists Code Example
- 5 elements in the list:
  - Element 1 - new Sheep()
  - Element 2 - new Cow("Clarabelle")
  - Element 3 - new Cow()
  - Element 4 - new Sheep("Dolly")
  - Element 5 - new Cow("Buttercup")
- Iterate through the list and call noise() for each element
- lines 142-145
  - Element 1 - sheep.noise() – line 98
    - Output: A Sheep says baa
  - Element 2 - cow.noise() – line 51
    - Output: Clarabelle the Cow says moo

## Revisit Lists Example

- All are Animal objects, when we iterate through list and call noise() for each animal
  - Output:

```
A Sheep says baa
Clarabelle the Cow says moo
A Cow says moo
Dolly the Sheep says baa
Buttercup the Cow says moo
```

- Line 144 - Simple but so powerful
- Objects are treated the same but result is different for each object
- Is this Polymorphism ? Why?

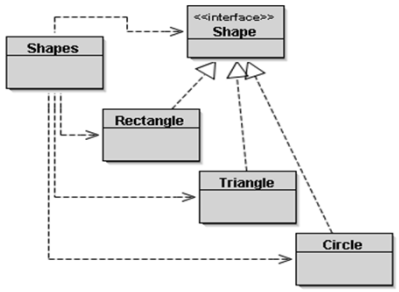
## Quick Quiz

- What is polymorphism?
- Give an example of polymorphism.

## Another Example of Polymorphism

- Consider a rectangle, a triangle and a circle.
  - They are all shapes
  - They all have a perimeter and an area
  - But the area and perimeter are calculated differently for each shape
  - Can we call the same methods on each shape?
  - area() ? perimeter() ?
  - Will the method behave differently for each shape?
  - Is this Polymorphism?

## Polymorphism Example



## Polymorphism Code Exercise

- Polymorphism Code Exercise – see handout
- Triangle, Rectangle and Circle all inherit from Shape interface
- All supply implementation for area() and perimeter() methods

## Exercise

- Fill in the missing code:
  - lines 139-141
    - Add an instance of Rectangle to shapes
    - Add an instance of Circle to shapes
    - Add an instance of Triangle to shapes
  - lines 151-152
    - Call area() on this variable
    - Call perimeter() on this variable

## Answers

- lines 139-141

```
shapes.add(new Rectangle(5, 2));
shapes.add(new Circle(2));
shapes.add(new Triangle(3, 4, 5, 3));
```

- lines 151-152

```
System.out.println("Area is:" + current.area());
System.out.println("Perimeter is:“ + current.perimeter());
```

## Polymorphism

- A list of shape objects, iterate through the list, call area() and perimeter() for each shape
  - **Lines 151 – 152**

```
System.out.println("Area is:" + current.area());
System.out.println("Perimeter is:" + current.perimeter());
```

**The power of Polymorphism, Encapsulation and Inheritance**

Objects are treated the **same,** the results are **different** because each sub class has a **different implementation of area() & perimeter()**

## Dynamic Dispatch

- In the polymorphism example
  - LinkedList was declared as type Shape – line 133

```
private LinkedList<Shape> shapes = new LinkedList<Shape>();
```

  - current was declared as type Shape – line 149

```
for(Shape current : shapes)
```

- How did the program know to use the Rectangle, Circle and Triangle implementations (respectively) when it called area() and perimeter() on each element of the List?

## Dynamic Dispatch

- Remember that each element in the list was declared type Shape.
- Dynamic Dispatch – the dynamically allocated type (ie the actual type of the object) e.g. Rectangle, Triangle or Circle, determines which method of area() and perimeter() is called.
- **The runtime uses the dynamic type to determine which method call should be dispatched**
  - If the object is a Rectangle, the rectangle implementation of the area() or perimeter() method will be called.

## Quick Quiz

- What is Dynamic Dispatch ?

## Unique Child Features (non-polymorphism)

- If a child class has a unique feature not define by the parent, it can only be accessed using an if or case statement.
- This is **ugly** and should be avoided where possible. It can be confusing for other developers to understand and hard to maintain.
- But …. here is an example for the child method diameter() for the Circle class
- Unique Code Example – see handout

## Unique Child Features

- New method diameter() to add to the Circle class - Lines 2 – 8

```
public double diameter()
  {
    return 2 * radius;
  }
```

- Altered listProperties() method in Shapes class – Lines 10 - 28
- Actual code changed in listProperties() - Lines 20 – 23

```
if (current instanceof Circle)
{
    System.out.println("Diameter is:" +
  ((Circle)current).diameter());
}
```

- Where is the **UGLY** piece of code?

## Unique Child Features

- Unique Child Features - Lines 20 - 23
- if statement to test for Circle object

```
if (current instanceof Circle)
{
   System.out.println("Diameter is:" +
  ((Circle)current).diameter());
 }
```

- Call to diameter() method – note the cast to Circle first
  - Why?

## instanceof

- Please note: all lowercase, it is instanceof; not instanceOf
- instanceof tests the dynamic type of an object at runtime. That is, the actual type of the object at runtime.
- if (current instanceof Circle) – is testing if the object current has an actual type of Circle.
- In the case of this example the actual types are:
  - Rectangle, Triangle & Circle, depending on which object is current

## Summary: Polymorphism

- The term polymorphism refers to the practice of treating objects (of different types) the same; that is, they all supply the same interface.
- The differences are hidden inside the methods: each class offers the same methods, but the implementation is different for each class.
- We can declare the type of the list to be the parent type. We can then store any object of the parent class, or of a child class, in the list.

## Project Costs

- Hardware costs are ~ 20%
- Software development is ~ 20%
- Software modification is ~ 60%
- The ability to change and extend code easily is essential
- We must design and code for the next person
  - It is unlikely the person who wrote the initial code is still be there for modification phase

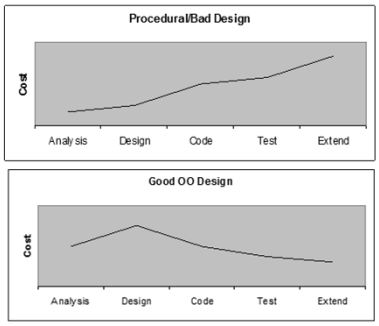
## Questions to ask

- How can I design a system that is
  - Easy to understand?
  - Easy to modify?
  - Easy to reuse?
- If there is a conflict between efficiency and reuse, reuse wins almost every time
- Efficiency can be added later if it is really needed ie database operations

## Benefits of good design

- Most time (and money) is spent extending existing code.
- Reuse is the key to a good OO design,
  - More effort at the start
    - Hard to design for reuse
  - Less effort in the end
    - Easy to reuse and extend

## Typical Cost/Phase Charts







## Capability Maturity Method (CMM)

- CMM measures the reliability (maturity) of software development processes. There are 5 levels:
  - Initial (chaotic, ad hoc)
    - No management practices or plan. Results are unpredictable
  - Repeatable
    - Practices can be used repeatedly, with predictable outcomes.
  - Defined
    - The process is defined and managed. New products can be reliably planned
  - Managed
    - Process is managed according to the metrics described in the Defined stage. Processes can be improved ad hoc.
  - Optimised
    - Active process management including deliberate, planned process optimisation and improvement.