

COMP 6699 – Object Oriented Programming

Week 10

**Polymorphism,
Final and Abstract Classes
and
Interfaces**

RECAP

- Understand the concept of inheritance and how it could be depicted in a class diagram and how it is implemented in a Java program
- Inheritance and methods [Override, Inherit, Add]
- Inheritance and fields/instance variables [Inherit, Add]
- Instance vs. Class variables
- Access control levels and recommended access levels
- The Cosmic superclass : Object (toString, equals, clone)
- Overloading and Overriding
- Inheritance : Applied (Person, Student, Teacher class diagram)
- Forum (Genus and Species)

Session Learning Outcomes

Upon completion of this session, students are expected to be able to

- Understand and apply the concept of Polymorphism
- Understand the concept of Final and Abstract classes and how to implement it
- Understand the concept of Interfaces and how it is being implemented
- Decide when to use Abstract classes or Interfaces in class design

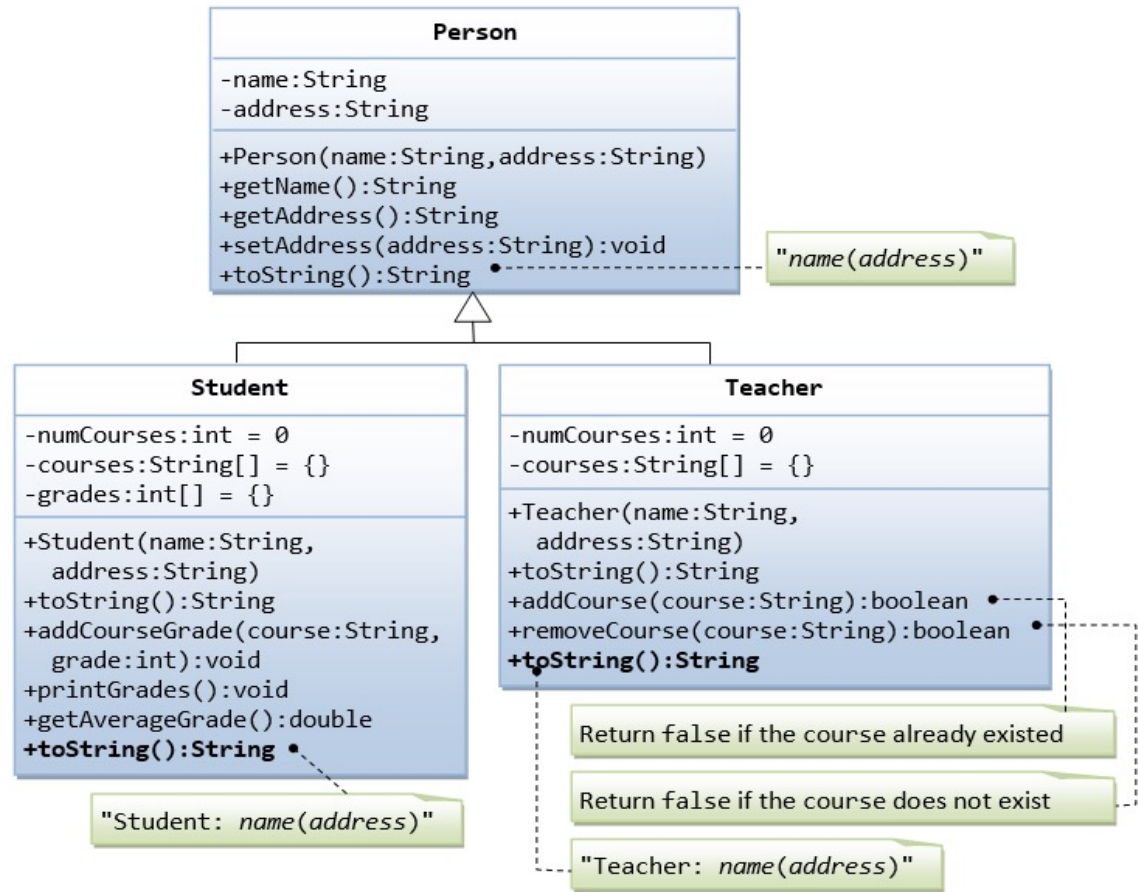


Polymorphism

- Polymorphism
 - The ability of a reference variable to change behavior according to what object it is holding.
 - This allows multiple objects of different subclasses to be treated as objects of a single superclass, while automatically selecting the proper methods to apply to a particular object based on the subclass it belongs to.
- To illustrate polymorphism, let us discuss an example.

Polymorphism

- Given the parent class Person and the subclass Student, we add another subclass of Person which is Teacher.



Polymorphism

- In Java, we can create a reference that is of type superclass to an object of its subclass. For example,

```
public static main( String[] args ) {  
  
    Person ref;  
    Student studentObject = new Student();  
    Teacher teacherObject = new Teacher();  
  
    ref = studentObject; //Person reference points to a Student object  
    ...  
    ...  
    ref = teacherObject; //Person reference points to Teacher object  
}
```

Polymorphism

- Now suppose we have a `getName()` method in our superclass `Person`, and we override this method in both the subclasses `Student` and `Teacher`.

```
public class Student {  
    public String getName(){  
        System.out.println("Student Name:" + name);  
        return name;  
    }  
}
```

```
public class Teacher {  
    public String getName(){  
        System.out.println("Teacher Name:" + name);  
        return name;  
    }  
}
```

Polymorphism

- Going back to our main method, when we try to call the getName() method of the reference Person ref, the getName() method of the Student object will be called.
- Now, if we assign ref to a Teacher object, the getName() method of Teacher will be called.

Polymorphism

```
1  public static void main( String[] args ) {  
2      Person ref;  
3      Student studentObject = new Student();  
4      Teacher teacherObject = new Teacher();  
  
5      ref = studentObject; //Person ref. points to a Student object  
  
6      //getName() of Student class is called  
7      String temp=ref.getName();  
8      System.out.println( temp );  
  
9      ref = teacherObject; //Person ref. points to a Teacher object  
10  
11     //getName() of Teacher class is called  
12     String temp = ref.getName();  
13     System.out.println( temp );  
14 }
```

Polymorphism

- Another example that illustrates polymorphism is when we try to pass references to methods.
- Suppose we have a static method **printInformation** that takes in a Person reference as parameter.

```
public static printInformation( Person p ){  
    . . . .  
}
```

Polymorphism

- *We can actually pass a reference of type Teacher and type Student to the printInformation method as long as it is a subclass of the class Person.*

```
public static main( String[] args )
{
    Student      studentObject = new Student();
    Teacher      teacherObject = new Teacher();

    printInformation( studentObject );

    printInformation( teacherObject );
}
```

Final Classes

- Final Classes

- Classes that cannot be extended

- To declare final classes, we write,

```
public final ClassName{  
    . . .  
}
```

- Example:

```
public final class Person {  
    . . .  
}
```

- Other examples of final classes are your wrapper classes and Strings.

Final Methods and Classes

- Final Methods

- Methods that cannot be overridden
- To declare final methods, we write,

```
public final [returnType] [methodName] ([parameters]) {  
    . . .  
}
```

- Example

```
public final String getName(){  
    return name;  
}
```

- Static methods are automatically final.

Abstract Classes

- Abstract class
 - a class that cannot be instantiated.
 - often appears at the top of an object-oriented programming class hierarchy, defining the broad types of actions possible with objects of all subclasses of the class.

Abstract Classes

- Abstract methods
 - methods in the abstract classes that do not have implementation
 - To create an abstract method, just write the method declaration without the body and use the abstract keyword
- For example,

```
public abstract void someMethod();
```

Sample Abstract Class

```
public abstract class LivingThing {  
    public void breath(){  
        System.out.println("Living Thing breathing...");  
    }  
  
    public void eat(){  
        System.out.println("Living Thing eating...");  
    }  
  
    /**  
     * abstract method walk  
     * We want this method to be overridden by subclasses of  
     * LivingThing  
     */  
    public abstract void walk();  
}
```


Abstract Classes

- When a class extends the LivingThing abstract class, it is required to override the abstract method walk(), or else, that subclass will also become an abstract class, and therefore cannot be instantiated.
- For example,

```
public class Human extends LivingThing {  
  
    public void walk(){  
        System.out.println("Human walks...");  
    }  
  
}
```

Coding Guidelines

- Use abstract classes to define broad types of behaviours at the top of an object-oriented programming class hierarchy and use its subclasses to provide implementation details of the abstract class.

Interfaces

- An interface
 - is a special kind of block containing method signatures (and possibly constants) only.
 - defines the signatures of a set of methods, **without the body**.
 - defines a standard and public way of specifying the behavior of classes.
 - allows classes, regardless of their locations in the class hierarchy, to implement common behaviors.
 - NOTE: interfaces exhibit polymorphism as well, since program may call an interface method, and the proper version of that method will be executed depending on the type of object passed to the interface method call.

Why do we use Interfaces?

- *To have unrelated classes implement similar methods*
 - *Example:*
 - *Class Line and MyInteger*
 - *Not related*
 - *Both implements comparison methods*
 - *isGreater*
 - *isLess*
 - *isEqual*

Why do we use Interfaces?

- *To reveal an object's programming interface without revealing its class*
- *To model multiple inheritance which allows a class to have more than one superclass*

Creating Interfaces

- To create an interface, we write:

```
public interface [InterfaceName] {  
    //some methods without the body  
}
```

Implementing the Interface

```
class ACMEBicycle implements Bicycle {  
    // remainder of this class implemented as before  
}
```

Note: if your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

Creating Interfaces

- As an example, let's create an interface that defines relationships between two objects according to the “natural order” of the objects.

```
public interface Relation
{
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```


Creating Interfaces

- To use an interface, we use the **implements** keyword.
- For example,

```
/** This class defines a line segment */  
public class Line implements Relation {  
    private double x1;  
    private double x2;  
    private double y1;  
    private double y2;  
  
    public Line(double x1, double x2, double y1, double y2){  
        this.x1 = x1;  
        this.x2 = x2;  
        this.y1 = y1;  
        this.y2 = y2;  
    }  
}
```

//program continued in the next slide

Creating Interfaces

```
public double getLength(){
    double length = Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)* (y2-y1));
    return length;
}

public boolean isGreater( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen > bLen);
}

public boolean isLess( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen < bLen);
}

public boolean isEqual( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen == bLen);
}
}
```

Creating Interfaces

- When your class tries to implement an interface, always make sure that you implement all the methods of that interface, or else, you would encounter this error,

Line.java:4: Line is not abstract and does not override abstract method
isGreater(java.lang.Object,java.lang.Object) in Relation

public class Line implements Relation

^

1 error

Interface vs. Abstract Class

- *ALL Interface methods have no body*
- *Some Abstract classes have method with implementation*
- *An interface can only define constants*
- *An abstract class is just like an ordinary class that can declare variables*
- *Interfaces have no direct inherited relationship with any particular class, they are defined independently*
- *Abstract classes can be subclassed*

Interface vs. Class

- Common:

- Interfaces and classes are both types
- This means that an interface can be used in places where a class can be used
- For example:

```
PersonInterface    pi = new Person();  
Person             pc = new Person();
```

- Difference:

- You cannot create an instance from an interface
- For example:

```
PersonInterface    pi = new PersonInterface(); //ERROR!
```

Interface vs. Class

- Common:
 - Interface and Class can both define methods
- Difference:
 - Interface does not have any implementation of the methods

Extending Classes vs. Implementing Interfaces

- A class can only EXTEND ONE super class, but it can IMPLEMENT MANY interfaces.
- For example:

```
public class Person implements PersonInterface,  
                                LivingThing,  
                                WhateverInterface {  
  
    //some code here  
}
```


Inheritance among Interfaces

- Interfaces are not part of the class hierarchy. However, interfaces can have inheritance relationship among themselves
- For example:

```
public interface PersonInterface {  
    ...  
}
```

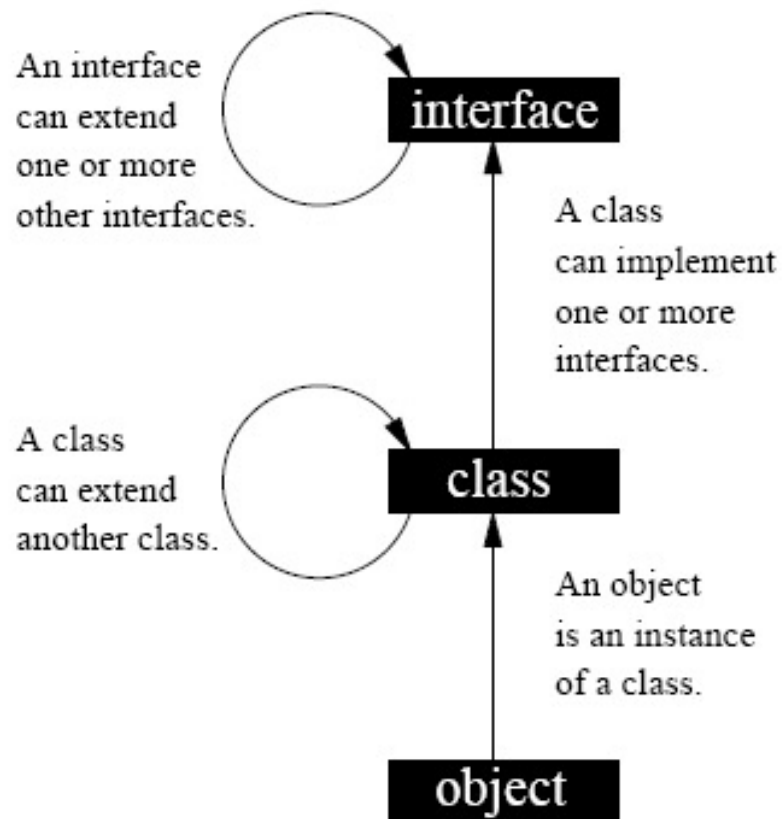
```
public interface StudentInterface extends PersonInterface {  
    ...  
}
```

Case Problem: Interfaces

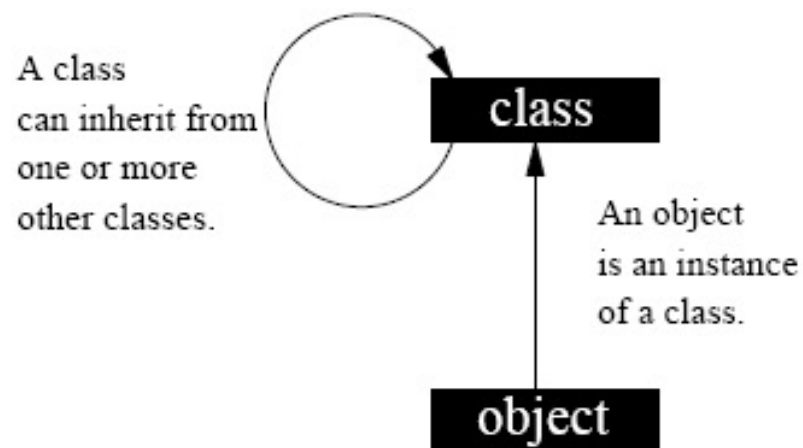
Exercise on Interfaces

- **By the end of this exercise you will be able to**

1. *Use interfaces as a solution to the problem of multiple inheritance.*
2. *Understand how an interface is similar to an abstract class with all methods abstract and no properties except static constants.*



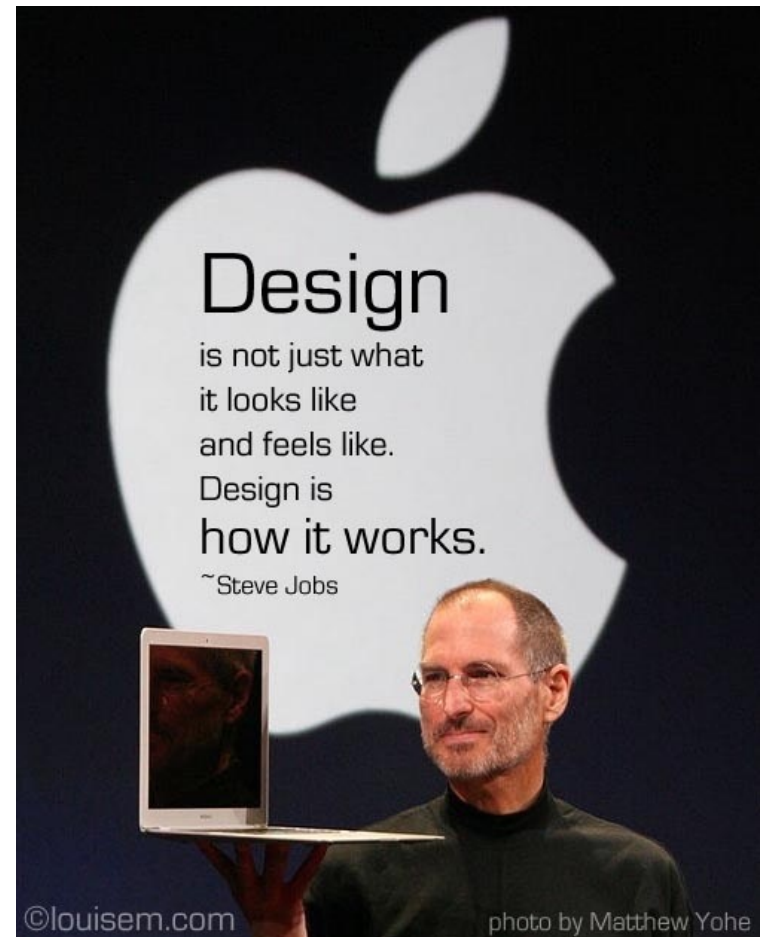
Java



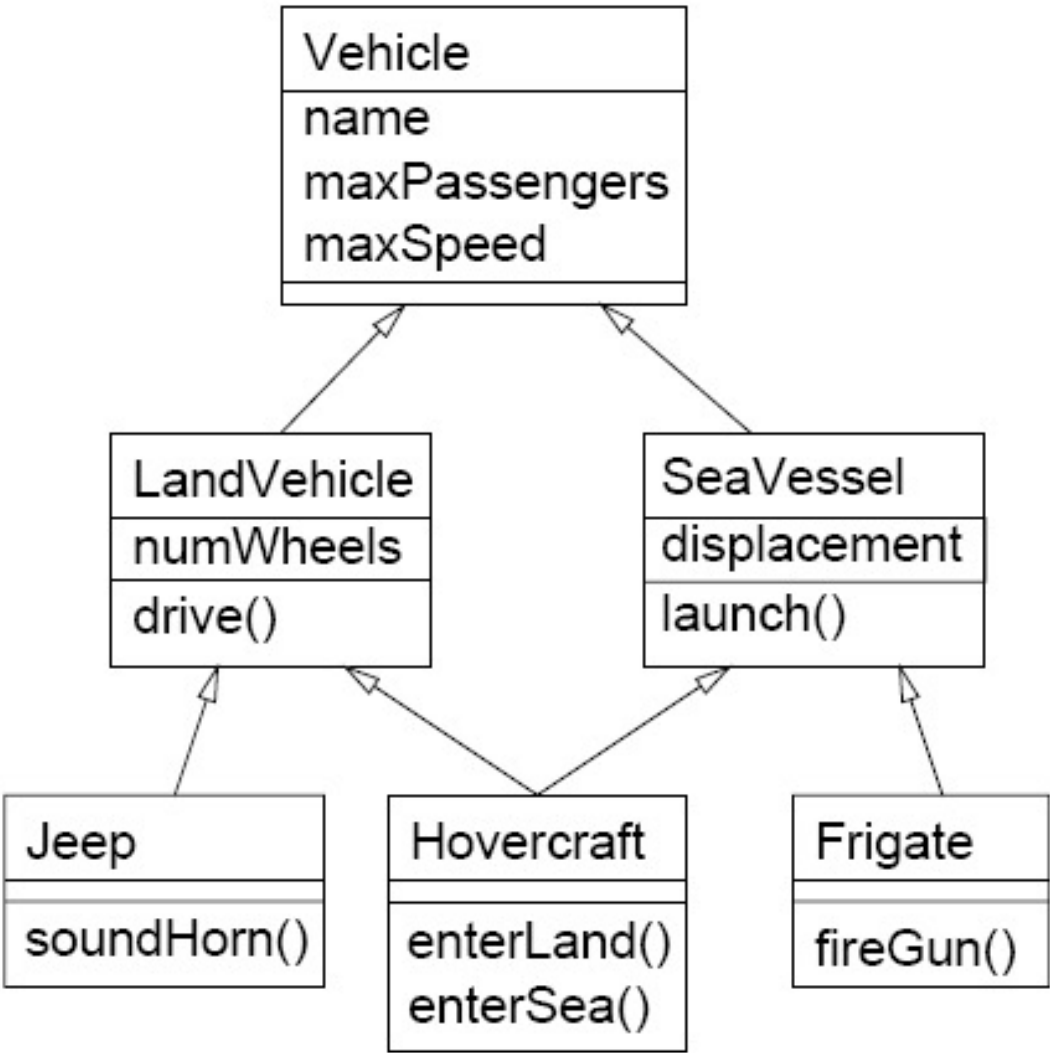
C++

Tasks

1. Analysis
2. Solution Design
3. Implementation



Problem



To Do

Implement the solution in a Java by:

1. Show the modified UML Classe Diagram Design
2. Implement in code and add the following functionalities:
 - a) By copying the pattern from the other interfaces, write an interface `IsEmergency` which extends no other interface and contains just one method `soundSiren` which takes no arguments and returns no value.
 - b) Write a class `PoliceCar` that implements the `IsEmergency` and `IsLandVehicle` interfaces.
 - c) In addition to the methods you have written for the `PoliceCar` class, think of a new method or property that police cars have and add it to the class.
 - d) Add the `PoliceCar` class and the `IsEmergency` interface to the new UML diagram. Show all methods and properties.
 - e) Construct a `PoliceCar` object and add it to the array/list `myArray/myList` in the main method.

End of Lecture...