Inheritance - summary

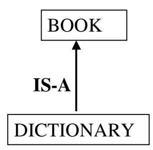
- Through inheritance a new class may be **derived** from an old one *(superclass).*
- The new class (*subclass)* automatically contains some or all of the variables and methods of the original class. The software designer can then add new variables and methods to the subclass, or modify the inherited ones to define the class appropriately.
- At the heart of inheritance is the idea of *software reuse* - using exiting software components to create new ones.

- Java uses the word **extends** to indicate that a new class is being derived from another.
- The derived class is a more specific version of the original. e.g.,

```
import Book;
public class Dictionary extends Book{
. . .
}
```


Inheritance - summary

- A variable or method declared with **protected** visibility may be accessed by a class in the same package.
- Inherited variables and methods retain the effect of their original visibiliity modifiers.
- Constructors are not inherited although they have public visibility.
- The word **super** is used to refer to a parent class.




```
protected in pages;
public void pageMessage() { …};

//inherits from book:
pages and pageMessage();

//own variables/methods;
private int definitions;
public Dictionary(int num1, int defn);
public void definitionMessage() { …};
```


Access Modifiers

- The child class inherits:
  - public variables and methods
  - protected variables and methods
  - protected variables and methods can be accessed by any class in the same package
    - » The **String class** is part of **java.lang** package
    - » other packages:

      java.applet, java.math, java.io, java.util

super( ) reference

```
public class Dictionary extends Book{
  private int definitions;

public Dictionary(int numpages, int defn)
{
  super(numpages);
  definitions = defn;
}
. . .
}
```

Overriding methods

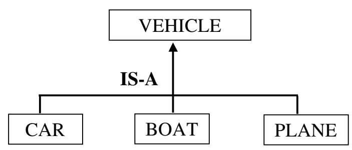
- When a child class defines a method with the same name and signature as a method in the parent class, we say the child's version *overrides* the parent's version.
  - When println() is called with an **object** parameter, toString is called to determine what to print.
  - The toString() method is defined in the class Object and overriden in the String class.
- A method can be defined with the **final** modifier so that a child class cannot override it ensuring a child class uses a particular definition for a method.

Abstract Classes

- a generic concept of a class
  - cannot be instantiated
  - contains one or more abstract methods, which have no definition
  - Unlike *Interfaces* an abstract class can contain methods which are not abstract and data declarations other than constants.

- The class uses the modifier *abstract* in the class header.
- Any class that contains one or more *abstract* methods must be declared as *abstract.*
- The *abstract* modifier must be applied to to each abstract method.
- An *abstract* class represents and entity that is too ill defined to be useful by itself. It contains partial descriptions inherited by its descendants.

Class Hierarchy



Abstract Classes

- Usually, the child class will provide a specific definition for an abstract method inherited from its parent (i.e., it overrides the method).
- If a child class does not give a definition for every inherited abstract method it too is considered an abstract class.
- *Abstract* methods should not be modified as *final* or *static*.

Polymorphism

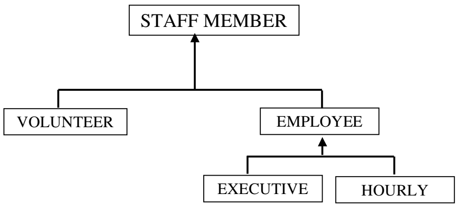
- An object declared to refer to a particular class can be used to refer to an object of *any* class to which it is related by inheritance.

```
Employee new1;
Executive new2 = new Executive();
new2 = new2;
```

- An **Object** reference can be used to refer to to any object - all classes are descendants of **Object.**

Polymorphism

- **new1** can be polymorphic because at any time it could refer to an Employee object or to an Executive object.



Professor D. Vernon, Department of Computer Science, National University of Ireland, Maynooth