Object Oriented Programming

- *Inheritance*

Single Inheritance

- The Java model of programming makes extensive use of Inheritance
- Normal inheritance plays two roles in programming.
  - When class B inherits from class A, it “reuses” all the non-private methods and members of class A.
  - B also becomes a subtype of A.

- Inheritance Hierarchies
  - The standard way of drawing out inheritance is through a tree-like hierarchy.
  - In UML the arrows point from the subclass to the superclass. This is because the superclass doesn't generally know of all of its subclasses but the subclasses know of the superclass.

- Inheritance for Code Reuse
  - The first side effect of inheritance is gaining “copies of” non-private members.
  - This means that if A had a public method get() then B will also have a public method get().

- Virtual methods
  - One of the powers of Java is that you don't always have to use the methods defined by the superclass. You can override them in the subclass.
  - Methods that can be overridden are called virtual methods. By default all methods in Java are virtual, which means they can all be overriden.

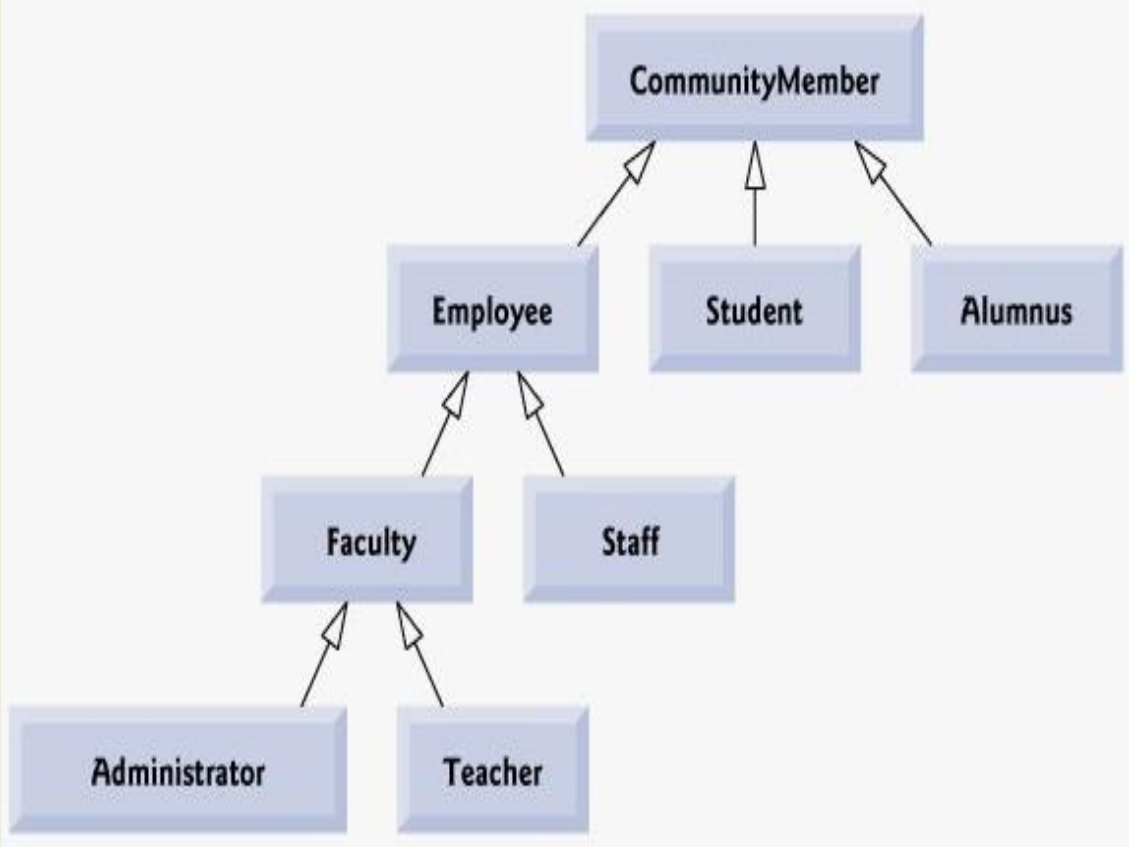
- Inheritance for Subtyping
  - Inheritance also provides subtyping. This is because the subclass has all the public methods and members of the superclass.
  - Formally, when we say that B is a subtype of A, what we are saying is that any place in the code where an A is expected, a B can be used, or a B can always take the place of an A.

- Single Inheritance of Classes
  - Java only allows single inheritance of classes i.e. a class can only inherit from one superclass
  - This greatly simplifies code by reducing ambiguity. C++ has multiple inheritance which causes one to frequently need to specify which superclass of a given class a method should be called through.

Inheritance examples

Superclass	**Subclasses**
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle
Loan	CarLoan, HomeLoan, MortgageLoan
Employee	Faculty, Staff
BankAccount	CurrentAccount, SavingsAccount

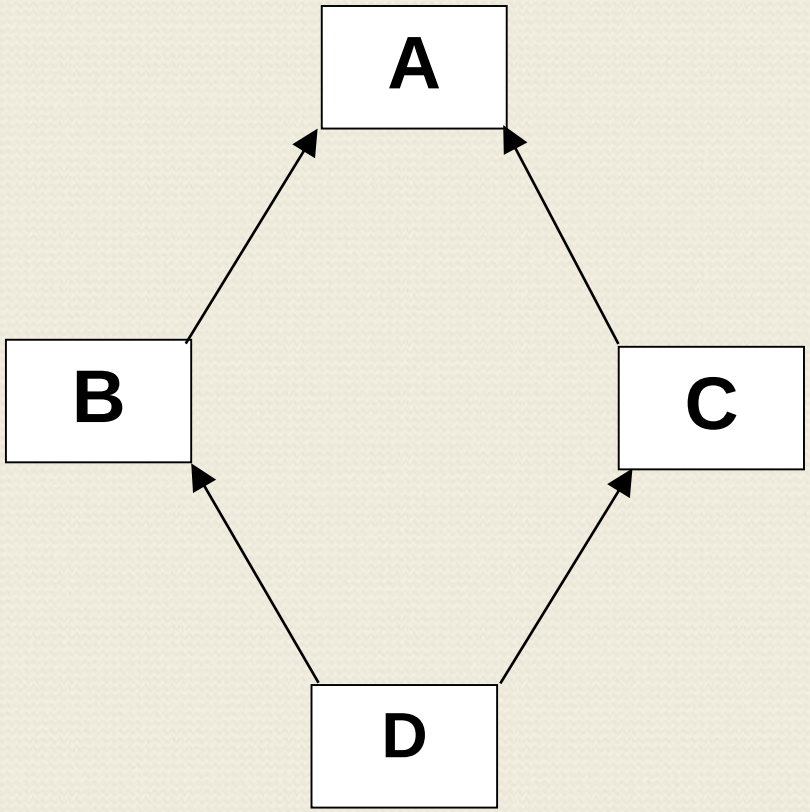
Sample UML single inheritance hierarchy



Multiple inheritance

- Java uses interfaces instead of pure multiple inheritance
- Classes in Java can only inherit from one single class (single inheritance) but can inherit from multiple interfaces
- This prevents the multiple inheritance problem where for example two superclasses of class D inherit from A resulting in conflicts in class D, as shown in figure below.

Fig: multiple inheritance problem



Defining interfaces

- We use the **`interface`** keyword to define an interface e.g.

```
public interface InterfaceName
{
    variable declaration;
    methods declaration;
}
```

- An interface is a kind of a class
- The difference is that interfaces only define abstract methods and final fields i.e. data fields are all constants and methods don't have a definition
- It is the responsibility of the class that implements an interface to define the code for the methods

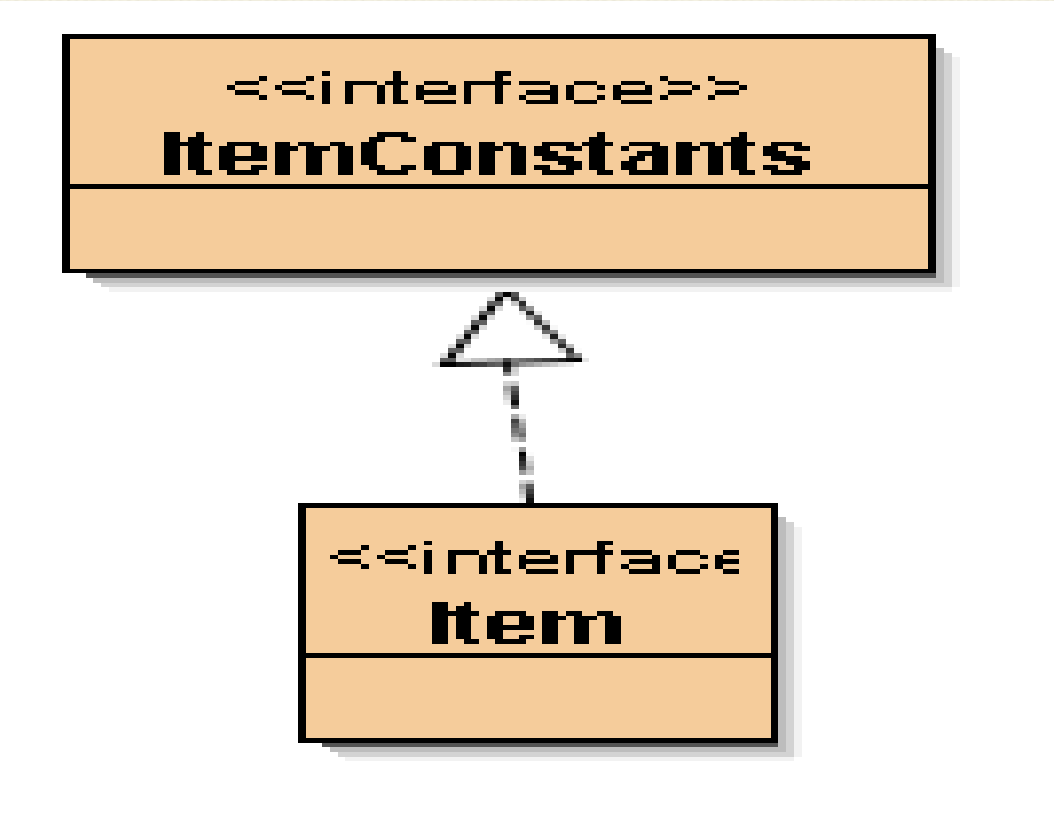
- Example

```
public interface Item
{
    static final int code = 1001;
    float compute(float x, float y);
    void show();
}
```

Extending interfaces

- Just like classes, interfaces can be extended
- The only difference is that the extending class (subinterface) must be an interface and not an ordinary class
- The new subinterface will inherit all members of the superinterface
- We use the keyword extends to subinterface an interface e.g.

Example



```
public interface ItemConstants
{
    int code = 1001;
    String name = "Fan";
}
public interface Item extends ItemConstants
{
    void display();
}
```

- NB:
  - Although keywords final and static are absent, fields in the two interfaces above are treated as constants
  - Subinterfaces cannot implement inherited methods because they are still interfaces

Implementing interfaces

- We inherit properties of interfaces into classes as follows

```
class classname implements interfacename
{
    body of classname
}
```

- Using interfaces can make our shapes program a lot easier e.g.

```
public interface Area {
    final static float pi = 3.14F;
    float compute(float x, float y);
}
class Rectangle implements Area {
    public float compute(float x, float y){
        return(x*y);
    }
}
```

```
public class Circle implements Area
{
    public float compute(float x, float y)
    {
        return(pi*x*x);
    }
}
```

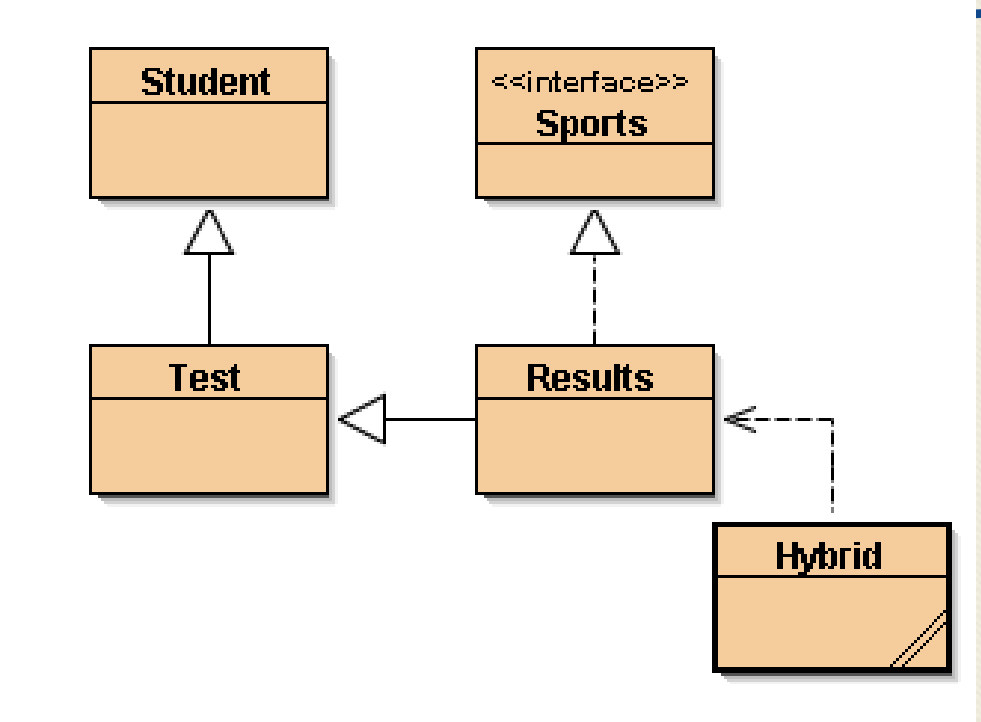
```
public class InterfaceTest {
    public static void main(String args[]) {
        Rectangle rect = new Rectangle();
        Circle circle = new Circle();
        Area area;
        area = rect;
        System.out.println("Area of Rectangle = "+area.compute(10,20));
        area = circle;
        System.out.println("Area of Circle = "+area.compute(10,0));
    }
}
```

Accessing interface variables

- You can directly access interface variables from a class e.g.

```
interface A {
     int m = 10;
     int n  = 50;
}
class B implements A {
     int x = m;
     void methodB(int size) {
          …
          if(size<n)
          …
       }
}
```

Case Study: Student System



```
class Student
{
    int rollNumber;
    void getNumber(int n)
    {
        rollNumber = n;
    }
    void putNumber()
    {
        System.out.println("Roll No: "+rollNumber);
    }
}
```

```
class Test extends Student {
    float part1, part2;
    void getMarks(float m1, float m2) {
        part1 = m1;
        part2 = m2;
    }
    void putMarks() {
        System.out.println("Marks obtained ");
        System.out.println("Part 1 = "+part1);
        System.out.println("Part 2 = "+part2);
    }
}
```

```
interface Sports
{
    float sportWt = 6.0F;
    void putWt();
}
```

```
class Results extends Test implements Sports {
    float total;
    public void putWt() {
        System.out.println("Sports Weight = "+sportWt);
    }
    void display() {
        total = part1 + part2 + sportWt;
        putNumber();
        putMarks();
        putWt();
        System.out.println("Total score = "+total);
    }
}
```

```
class Hybrid
{
    public static void main(String args[])
    {
        Results student1 = new Results();
        student1.getNumber(1234);
        student1.getMarks(27.5F, 33.0F);
        student1.display();
    }
}
```

Output

Roll No: 1234

Marks obtained

Part 1 = 27.5

Part 2 = 33.0

Sports Weight = 6.0

Total score = 66.5