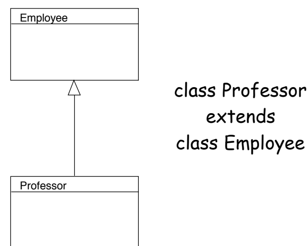
Inheritance

Inheritance

- "Inheritance" is a way of **extending** or building one class on top of another
  - The original class is called the **base class**, or **parent class**.
  - The new class is called the **derived class**, or **child class**.

Diagrammatic Notation (UML)



class Professor
extends
class Employee

UML = Unified Modeling Language

Inherited Capabilities

- **Extension**:
  The derived class can potentially use all data components and methods from the base class, and add more of its own.
- **Over-Riding:**
  It can also selectively re-define or "over-ride" methods of the same name. (It is a good idea to keep the same intuitive meaning of a method.)

Purposes of Inheritence

- **Reusing**: Use the *same* concepts and code for *many* classes (base-class concepts and code shared by derived classes):
  - Work economy: less stuff to implement
  - Intellectual economy : less stuff to understand
- **Generalizing**: Tie together similar classes:
  - Increases the utility of methods that **use** such classes.

Extension = Java Inheritance

- In Java, the keyword for "inherits from" is
  extends
- The derived class *extends* the base class.
- Extension allows over-riding as well; there is no separate keyword for over-riding.

Extension Example

- class `Account` defines a basic bank account
- class `CheckingAccount` defines a special account for check-writing

```
class Account
{
Money balance;

Account(Money initialBalance)
  {
  balance = initialBalance;
  }

void deposit(Money amount)
  {
  balance = balance.add(amount);
  }

boolean withdraw(Money amount)
  {
  if( balance.lessThan(amount) )
    return false;
  balance = balance.subtract(amount);
  return true;
  }

void showBalance(PrintStream out)
  {
  out.println("Balance: " + balance);
  }
```

Only allow withdrawal if sufficient funds; return boolean to indicate success or failure.

```
class CheckingAccount extends Account
{
Money serviceCharge;

CheckingAccount(Money initialBalance, Money serviceCharge)
  {
  super(initialBalance);
  this.serviceCharge = serviceCharge;
  }

boolean cashCheck(Money amount)
  {
  return withdraw(amount.add(serviceCharge));
  }
```

(continued next page)

Additional variable for the derived class.

"super" means "the constructor of the base class".

Added service charge

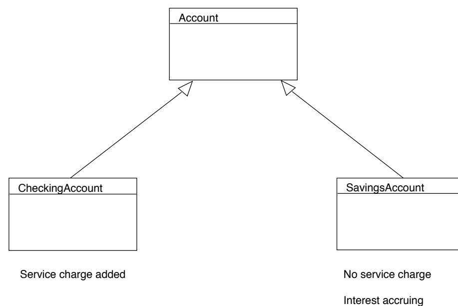
```
(program continued)

public static void main(String arg[])
{
CheckingAccount myCheckingAccount =
  new CheckingAccount(new Money(10000),
                      new Money(100));

myCheckingAccount.showBalance(System.out);

myCheckingAccount.deposit(new Money(5000));
myCheckingAccount.showBalance(System.out);

myCheckingAccount.cashCheck(new Money(2000));
myCheckingAccount.showBalance(System.out);

myCheckingAccount.cashCheck(new Money(1000));
myCheckingAccount.showBalance(System.out);

myCheckingAccount.withdraw(new Money(1000));
myCheckingAccount.showBalance(System.out);
}
}
```

Program Output

```
Balance: $100.0
Balance: $150.0
Balance: $129.0
Balance: $118.0
Balance: $108.0
```

(Link to the complete program: **Bank**)

Multiple derived classes

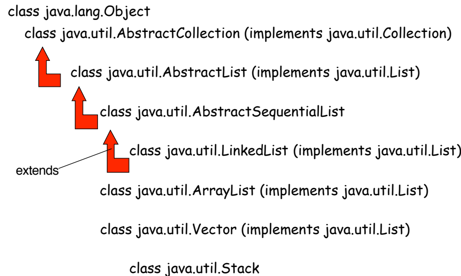


Which methods can be over-ridden?

- In order to be over-ridden, a method must be declared either:
  - **public**
  - **protected**

in the base class.

Inheritance Examples abound in Java Libraries



Implications of Inheritance

- The preceding diagram means, for example, that to find all methods for class java.util.Stack, you may wish to look at method descriptions in:
  - java.util.Vector
  - java.util.AbstractList
  - java.util.AbstractCollection
  - java.lang.Object

Methods of java.util.Stack

- Methods of Stack proper:
  - Object push(Object item)
  - Object pop()
  - boolean empty()
  - Object peek()
  - int search(Object o)
- Methods of Vector:
  - add, add, addAll, addAll, addElement, capacity, clear, clone, contains, containsAll, copyInto, elementAt, elements, ensureCapacity, equals, firstElement, get, hashCode, indexOf, insertElementAt, isEmpty, lastElement, lastIndexOf, remove, removeAll, removeAllElements, removeElement, removeElementAt, removeRange, retainAll, set, setElementAt, setSize, size, subList, toArray, toString, trimToSize
- Methods of AbstractList:
  - iterator, listIterator
- Methods of Object (not otherwise over-ridden)
  - finalize, getClass, notify, notifyAll, wait

Testing where Object is in Hierarchy

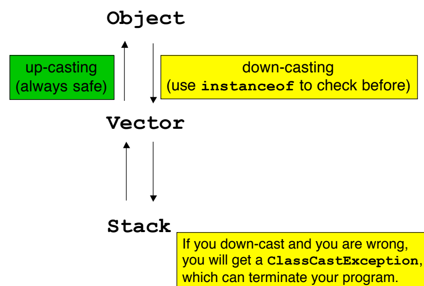
- *instanceof* operator

  Object ob = …;

  if(ob instanceof Vector) …
  if(ob instanceof Stack) …

More than one can be true!

Casting



class *Object*

- Object is the ancestor of all classes
- Some methods of Object:
  - boolean equals(Object)
  - Class getClass(): returns the “most derived” *Class object*
  - String toString()
- Method of class Class:
  - String getName()
  - So Ob.getClass().getName() will get you the class name of the object.

Implementing an Interface is similar to Inheritance

- Interface ≈ Base Class
- Implementation ≈ Derived Class

- By declaring methods to use the Interface rather than the Implementation class as an argument, more **generality** is afforded to that method.

Example

- `java.util.Iterator` is standard interface
- Make `ClosedList.Iterator` implement `java.util.Iterator`
- Any other code accepting a `java.util.Iterator` can now use our:
  `ClosedList.Iterator`
- We can *still* do everything we did before.

Example (a05)

- There are many ways to implement a Queue data structure:
  - Singly-linked list
  - Doubly-linked list
  - Array
  - Circular array
- Choices may be governed by usage patterns.
- Therefore:
  - Define an interface for Queue
  - Provide one or more implementations
  - Example: SLQueue: Use singly-linked list.

Multiple Inheritance

- Some languages allow one class to derive from **multiple** base classes; Java does not.

- The nearest thing would be a class deriving from a **single** basic class and implementing one or more interface at the same time.