Inheritance

Version 3.3 - March 2014

Inheritance

- A class can be a sub-type of another class
- The derived class contains
  - all the members of the class it inherits from
  - plus any member it defines explicitly
- The derived class can override the definition of existing methods by providing its own implementation
- The code of the derived class consists of the changes and additions to the base class

Addition

```
class Employee{
   String name;
   double wage;
   void incrementWage(){…}
}
```

```
class Manager extends Employee{
   String managedUnit;
   void changeUnit(){…}
}
```

```
Manager m = new Manager();
m.incrementWage(); // OK, inherited
```

Overriding

```
class Vector{
   int vect[];
   void add(int x) {…}
}
```

```
class OrderedVector extends Vector{
   void add(int x){…}
}
```


Inheritance and polymorphism

```
class Employee{
 private String name;
 public void print(){
   System.out.println(name);
 }
}
class Manager extends Employee{
   private String managedUnit;

   public void print(){ //overrides
     System.out.println(name); //un-optimized!
     System.out.println(managedUnit);
   }
}
```

```
Employee e1 = new Employee();
Employee e2 = new Manager();
e1.print(); // name
e2.print(); // name and unit
```

Inheritance and polymorphism

```
Employee e1 = new Employee();
Employee e2 = new Manager(); //ok, is_a
e1.print(); // name
e2.print(); // name and unit
```

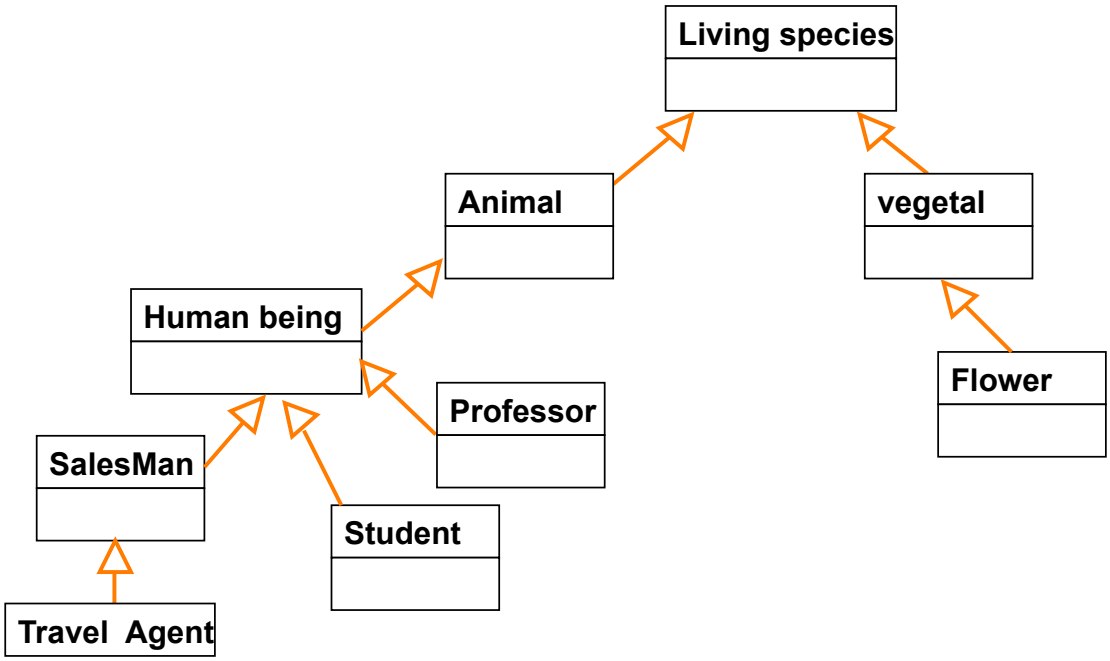

Why inheritance

- Frequently, a class is merely a modification of another class. In this way, there is minimal repetition of the same code
- Localization of code
  - Fixing a bug in the base class automatically fixes it in the subclasses
  - Adding functionality in the base class automatically adds it in the subclasses
  - Less chances of different (and inconsistent) implementations of the same operation

Inheritance in real Life

- A new design created by the modification of an already existing design
  - The new design consists of only the changes or additions from the base design
- CoolPhoneBook inherits PhoneBook
  - Add mail address and cell number

Example of inheritance tree



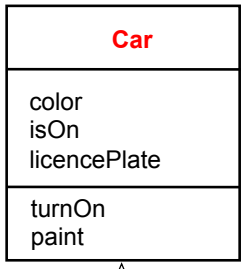
Inheritance terminology

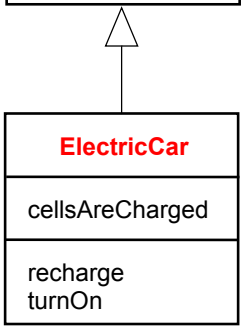
- Class one above
  - Parent class
- Class one below
  - Child class
- Class one or more above
  - Superclass, Ancestor class, Base class
- Class one or more below
  - Subclass, Descendent class

Inheritance in a few words

- Subclass
  - Inherits attributes and methods
  - Can modify inherited attributes and methods (override)
  - Can add new attributes and methods

Inheritance in Java: *extends*






```
class Car {

  String color;
  boolean isOn;
  String licencePlate;

  void paint(String color) {
    this.color = color;
  }

  void turnOn() {
    isOn=true;
  }
}
```

```
class ElectricCar extends Car
{

  boolean cellsAreCharged;

  void recharge() {
    cellsAreCharged = true;
  }

  void turnOn() {
    if(cellsAreCharged )
      isOn=true;
  }
}
```

SOftEng http://softeng.polito.it

ElectricCar

- Inherits
  - attributes (color, isOn, licencePlate)
  - methods (paint)
- Modifies (overrides)
  - turnOn()
- Adds
  - attributes (cellsAreCharged)
  - Methods (recharge)

SOftEng http://softeng.polito.it

Visibility (scope)

Example

```
class Employee {
  private String name;
  private double wage;
}

class Manager extends Employee {

  void print() {
    System.out.println("Manager" +
                       name + " " + wage);
  }
}
```

Not visible

Protected

- Attributes and methods marked as
  - public are always accessible
  - private are accessible within the class only
  - protected are accessible within the class and its subclasses

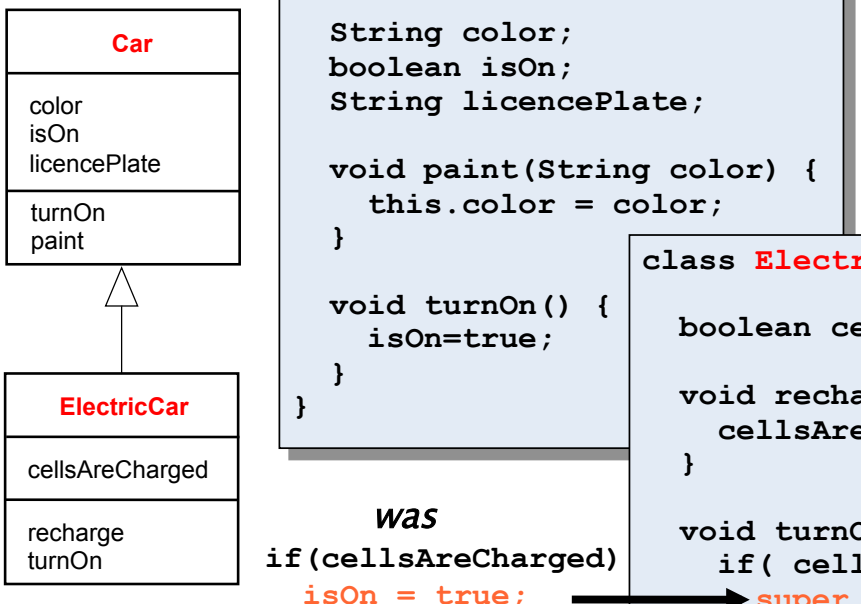
In summary

	Method in the same class	Method of another class in the same package	Method of subclass	Method of class in other package
`private`	✓			
package	✓	✓		
`protected`	✓	✓	✓	
`public`	✓	✓	✓	✓

Super (reference)

- "`this`" is a reference to the current object
- "`super`" is a reference to the parent class

Example




```
class Car {

  String color;
  boolean isOn;
  String licencePlate;

  void paint(String color) {
    this.color = color;
  }

  void turnOn() {
    isOn=true;
  }
}
```

was

```
if(cellsAreCharged)
  isOn = true;
```

```
class ElectricCar extends Car{

  boolean cellsAreCharged;

  void recharge() {
    cellsAreCharged = true;
  }

  void turnOn() {
    if( cellsAreCharged )
      super.turnOn();
  }
}
```

Attributes redefinition

```
Class Parent{
   protected int attr = 7;
}
Class Child{
   protected String attr = "hello";

   void print(){
     System.out.println(super.attr);
     System.out.println(attr);
   }

   public static void main(String args[]){
      Child c = new Child();
      c.print();
   }
}
```

Inheritance and constructors

Construction of child's objects

- Since each object "contains" an instance of the parent class, the latter must be initialized
- Java compiler automatically inserts a call to default constructor (no params) of parent class
- The call is inserted as the first statement of each child constructor

Construction of child objects

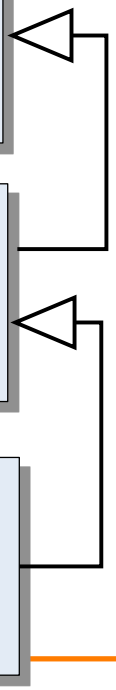
- Execution of constructors proceeds top-down in the inheritance hierarchy
- In this way, when a method of the child class is executed (constructor included), the super-class is completely initialized already

Example

```
class ArtWork {
  ArtWork() {
    System.out.println("ctor ArtWork"); }
}
```

```
class Drawing extends ArtWork  {
  Drawing() {
    System.out.println("ctor Drawing"); }
}
```

```
class Cartoon extends Drawing {
  Cartoon() {
    System.out.println("ctor Cartoon"); }
}
```



Example (cont'd)

```
Cartoon obj = new Cartoon();
```

```
ctor ArtWork
ctor Drawing
ctor Cartoon
```

A word of advice

- Default constructor “disappears” if custom constructors are defined

```
class Parent{
  Parent(int i){}
}
class Child extends Parent{ }
// error!
```

```
class Parent{
  Parent(int i){}
  Parent(){} //explicit default
}
class Child extends Parent { }
// ok!
```


Super

- If you define custom constructors with arguments
- and default constructor is not defined explicitly

➔ the compiler cannot insert the call automatically

Super

- Child class constructor must call the right constructor of the parent class, explicitly
- Use super() to identify constructors of parent class
- First statement in child constructors

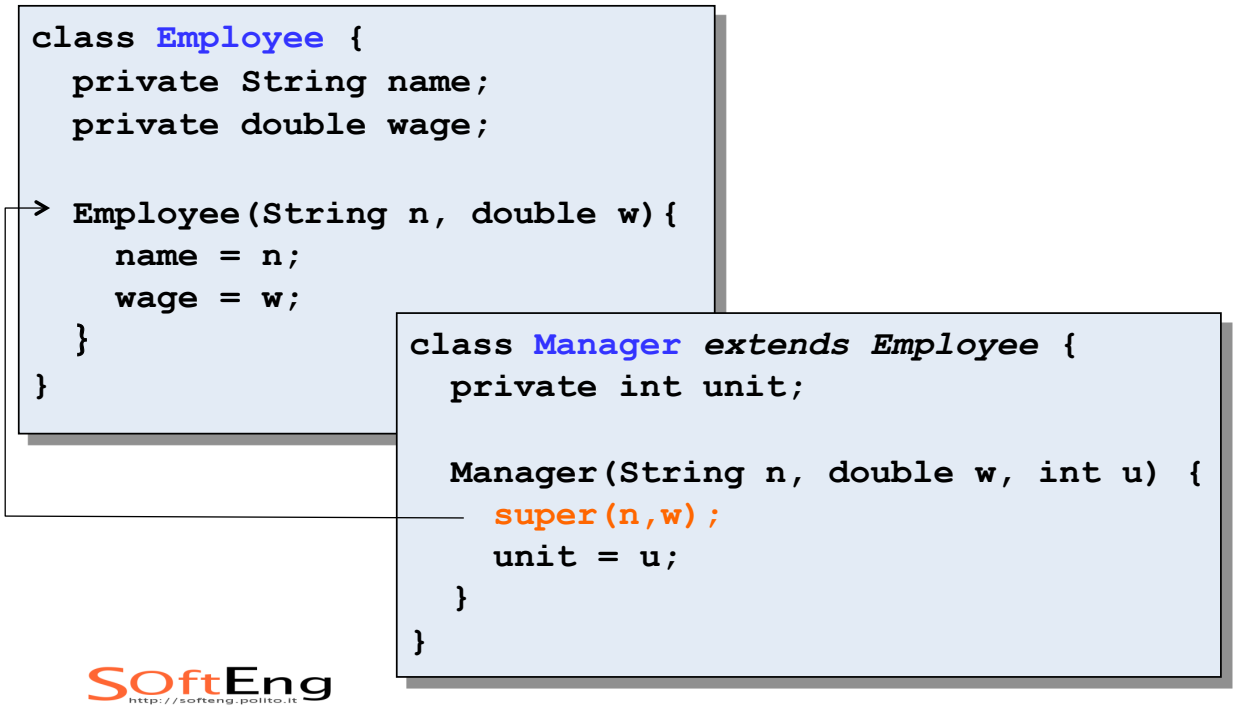
Example

```
class Employee {
  private String name;
  private double wage;
       ???
  Employee(String n, double w){
    name = n;
    wage = w;
  }
}
```

```
class Manager extends Employee {
  private int unit;

  Manager(String n, double w, int u) {
    super();  ERROR !!!
    unit = u;
  }
}
```

Example



```
class Employee {
  private String name;
  private double wage;

  Employee(String n, double w){
    name = n;
    wage = w;
  }
}
```

```
class Manager extends Employee {
  private int unit;

  Manager(String n, double w, int u) {
    super(n,w);
    unit = u;
  }
}
```


Depth of Inheritance Tree

- In general too deep inheritance trees put at risk the understandability of the code
  - An empirical limit is 5 levels

Dynamic binding/ polymorphism

Example

- `Car[] garage = new Car[4];`
- `garage[0] = new Car();`
- `garage[1] = new ElectricCar();`
- `garage[2] = new ElectricCar();`
- `garage[3] = new Car();`

- ```
  for(int i=0; i<garage.length; i++){
      garage[i].turnOn();
  }
  ```
  ```

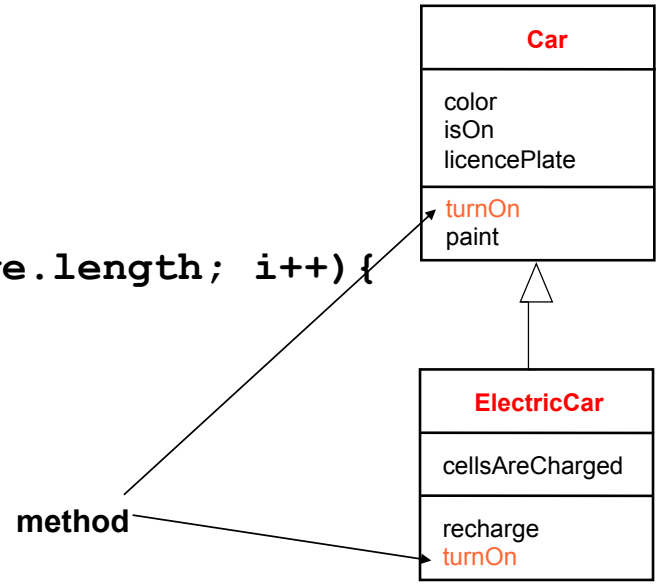
Binding

- Association message/method
- Constraint
  - Same signature

```
Car a;
for(int i=0; i<garage.length; i++){
    a = garage[i]
    a.turnOn();
}
```

message

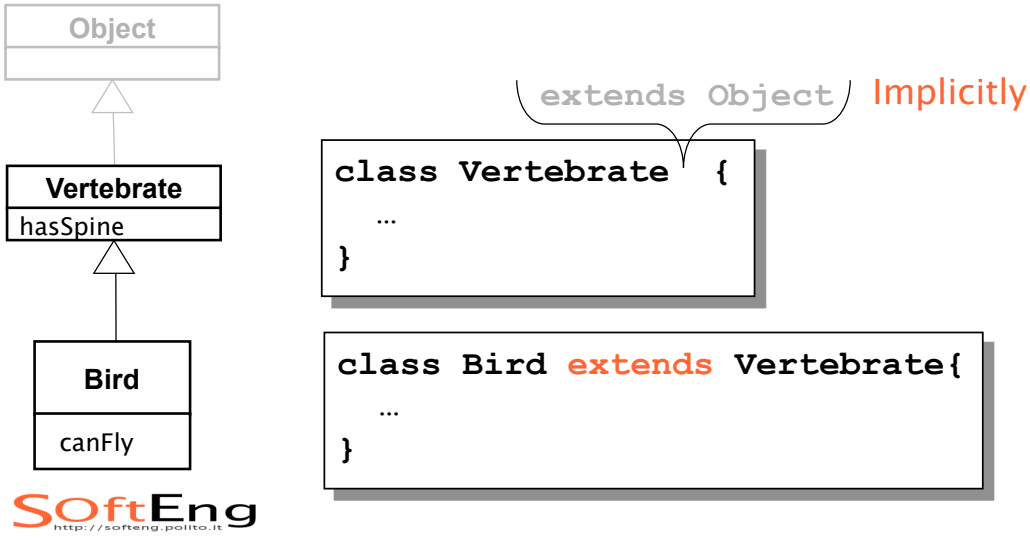
method



Object

Class Object

- **`java.lang.Object`**
- All classes are subtypes of Object



SOftEng http://softeng.polito.it

Class Object

- Each instance can be seen as an Object instance (see Collection)

- Class Object defines some services, which are useful for all classes
- Often, they are overridden in sub-classes

Object
toString() : String equals(Object) : boolean

SOftEng http://softeng.polito.it

Objects' collections

- References of type `Object` play a role similar to `void*` in C

```
Object [] objects = new Object[3];
objects[0] = "First!";
objects[2] = new Employee("Luca","Verdi");
objects[1] = new Integer(2);
for(Object obj : objects){
  System.out.println(obj);
}
```

Wrappers must be used instead of primitive types

SOftEng http://softeng.polito.it

Java Object

- `toString()`
  - Returns a string uniquely identifying the object
  - Default implementation returns:

    `ClassName@######`
  - Es:

    `org.Employee@af9e22`

Object
toString() : String equals(Object) : boolean

Java Object

- **`equals()`**
  - Tests equality of values
  - Default implementation compares references:

Object
toString() : String equals(Object) : boolean

```
public boolean equals(Object other){
return this == other;
}
```

  - Must be overridden to compare contents, e.g.:

```
public boolean equals(Object o){
Student other = (Student)o;
return this.id.equals(other.id);
}
```

SOftEng http://softeng.polito.it

System.out.print(Object)

- ***`print`*** methods implicitly invoke **`toString()`** on all object parameters

```
class Car{ String toString(){…} }

Car c = new Car();

System.out.print(c);  // same as...

... System.out.print(c.toString());
```

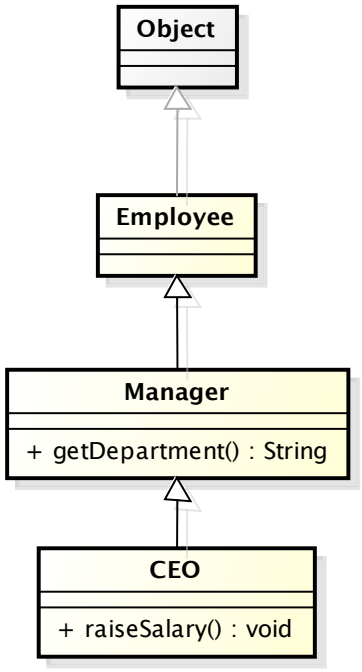
- Polymorphism applies when **`toString()`** is overridden

```
Object ob = c;

System.out.print(ob); // Car's toString() called
```

SOftEng http://softeng.polito.it

Company



Casting

Types

- Java is a strictly typed language, i.e., each variable has a type

- ```
  float f;
  f = 4.7;  // legal
  f = “string”; // illegal
  ```
- ```
  Car c;
  c = new Car();  // legal
  c = new String(); // illegal
  ```


Cast

- Type conversion (explicit or implicit)

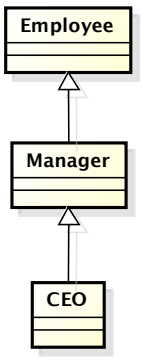
```
int i = 44;

float f = i;
// implicit cast 2c -> fp

f = (float) 44;
// explicit cast
```

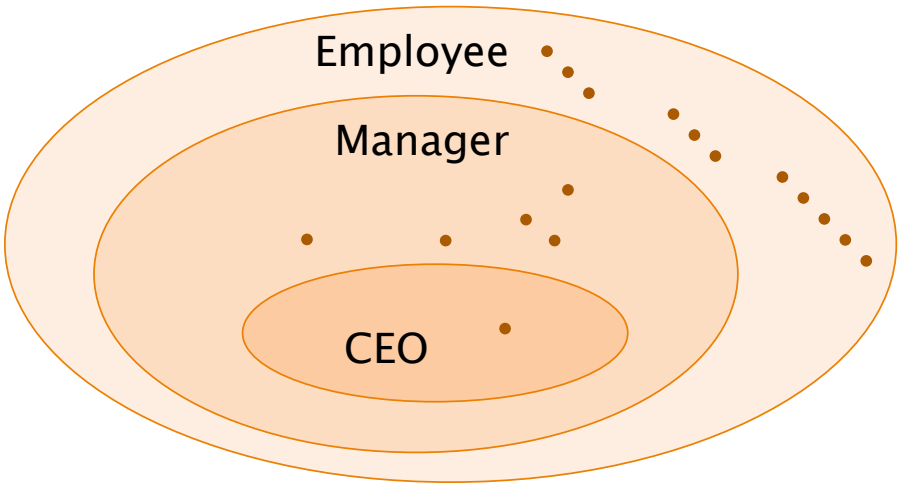
Cast – Generalization

- Things change slightly with inheritance
- Normal case…




```
Employee e = new Employee("Smith",12000);
Manager m = new Manager("Black",25000,"IT");
```


Generalization



Upcast

- Assignment from a more specific type (subtype) to a more general type (supertype)
  - `Employee e = new Employee(…);`
    `Manager m = new Manager(…);`
    `Employee em = m`
  - ∀ m ∈ Manager : m ∈ Employee
- Upcasts are always type-safe and are performed implicitly by the compiler
  - Though it is legal to explicitly indicate the cast

Upcast

- Motivation
  - You can treat indifferently object of different classes, provided they inherit from a common class

```
Employee[] team = {
  new Manager("Mary Black",25000,"IT"),
  new Employee("John Smith",12000),
  new Employee("Jane Doe",12000)
};
```

Cast

- Reference type and object type are distinct concepts
- A reference cast only affects the reference
  - In the previous example the object referenced to by '**em**' continues to be of Manager type
- Notably, in contrast, a primitive type cast involves a value conversion

Downcast

- Assignment from a more general type (super-type) to a more specific type (sub-type)
  - **`Manager mm = (Manager)em;`**
  - ∃ em ∈ Employee : em ∈ Manager
  - ∃ em ∈ Employee : em ∉ Manager
- Not safe by default, no automatic conversion provided by the compiler
  - MUST be explicit

Downcast

- Motivation
  - To access a member defined in a class you need a reference of that class type
    - Or any subclass

```
Employee emp = staff[0];
s = emp.getDepartment();
Manager mgr = (Manager)staff[0];
s = mgr.getDepartment();
```

Syntax Error: The method getDepartment() is undefined for the type Employee

Downcast – Warning

- The compiler trusts any downcast.
- JVM at run-time checks type consistency for all reference assignments

```
mgr = (Manager)staff[1];
```

ClassCastException: Employee cannot be cast to Manager

Down cast – safety

- Use the instanceof operator
  - **aReference instanceof aClass**
  - Returns true if the object referred to by lhs reference can be cast to the rhs class

```
if(staff[1] instanceof Manager){
     mgr = (Manager)staff[1];
}
```


Upcast to Object

- Each class is either directly or indirectly a subclass of **Object**
- It is always possible to upcast any instance to Object type (see Collection)

```
AnyClass foo = new AnyClass();
Object obj;
obj = foo;
```

Abstract classes

Abstract class

- Often, superclass is used to define common behavior for many child classes
- But the class is too general to be instantiated
- Behavior is partially left unspecified (this is more concrete than interface)

Abstract modifier

```
public abstract class Shape {

   private int color;

   public void setColor(int color){
      this.color = color;
   }

   // to be implemented in child classes
   public abstract void draw();
}
```

No method body

Abstract modifier

```
public class Circle extends Shape {

   public void draw() {
      // body goes here
   }
}


Object a = new Shape();  // Illegal: abstract
Object a = new Circle(); // OK
```

Interfaces

Java interface

- An interface is a special type of "class" where methods and attributes are implicitly public
  - Attributes are implicitly static and final
  - Methods are implicitly abstract (no body)
- Cannot be instantiated (no new)
- Can be used to define references

Purpose of interfaces

- Define a common “interface” that allows alternative implementations
- Provide a (set of) method(s) that can be called by algorithms
- Define a (set of) callback method(s)

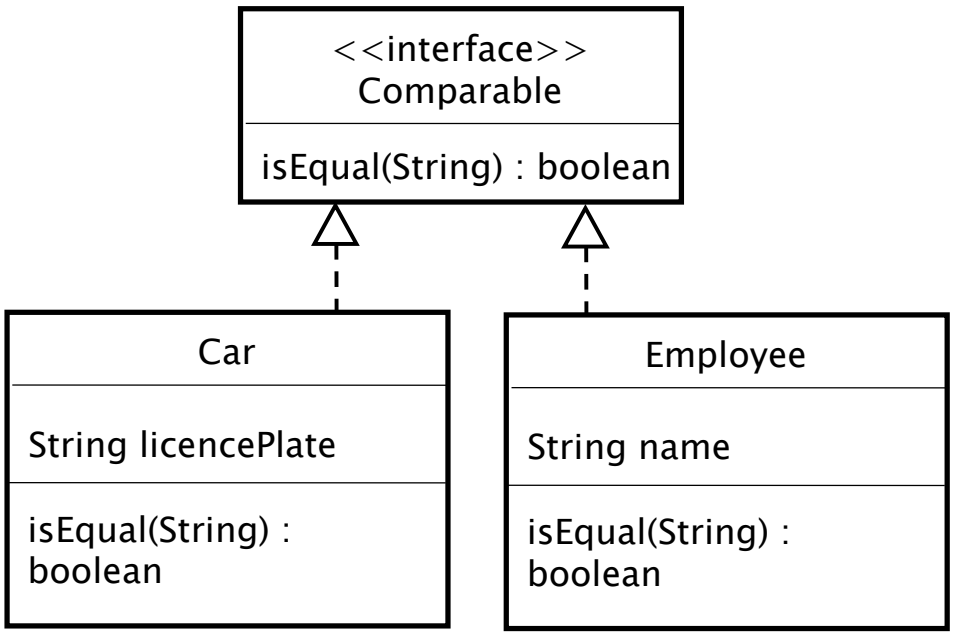
Alternative implementations

- Complex numbers

```
public interface Complex {
   double real();
   double imaginary();
   double modulus();
   double argument();
}
```

- Can be implemented using either Cartesian or polar coordinates

Example



Example (cont'd)

```
public interface Comparable {
    void isEqual(String s);
}
```

Public

```
public class Car implements Comparable {
  private String licencePlate;
  public void isEqual(String s){
    return licencePlate.equals(s);
  }
}
```

```
public class Employee implements Comparable{
  private String name;
  public void isEqual(String s){
    return name.equals(s);
  }
}
```

Example

```
public class Foo {
  private Comparable objects[];
  public Foo(){
    objects = new Comparable[3];
    objects[0] = new Employee();
    objects[1] = new Car();
    objects[2] = new Employee();
  }
  public Comparable find(String s){
    for(int i=0; i< objects.length; i++)
      if(objects[i].isEqual(s)
                        return objects[i];
  }
}
```


Rules (interface)

- An interface can extend another interface, cannot extend a class

```
interface Bar extends Comparable {
  void print();
}
```

interface

- An interface can extend multiple interfaces

```
interface Bar extends Orderable, Comparable{
  ...
}
```

interfaces

Rules (class)

- A class can extend only one class
- A class can implement multiple interfaces

```
class Person
   extends Employee
   implements Orderable, Comparable {…}
```


A word of advice

- Defining a class that contains abstract methods only is not illegal but..
  - You should use interfaces instead
- Overriding methods in subclasses can maintain or extend the visibility of overridden superclass's methods
  - e.g. protected int m() can't be overridden by
    - private int m()
    - int m()
  - Only protected or public are allowed

Homework

- See the doc of java.lang.Comparable

```
public interface Comparable{
  int compareTo(Object obj);
}
```

- Returns a negative integer, 0, or a positive integer as this object is less than, equal, or greater than `obj`

Homework (cont'd)

- Define Employee, which implements Comparable (order by ID)
- Define OrderedArray class
  - void add(Comparable c) //ordered insert
  - void print() //prints out
- Test it with the following main

Homework (cont'd)

```
public static void main(String args[]){

  int size = 3; // array size
  OrderedArray oa = new OrderedArray(size);

  oa.add( new Employee("Mark",   37645) );
  oa.add( new Employee("Andrew", 12345) );
  oa.add( new Employee("Sara",   97563) );

  oa.print();
}
```

Name

ID

Wrap-up session

- Inheritance
  - Objects defined as sub-types of already existing objects. They share the parent data/methods without having to re-implement
- Specialization
  - Child class augments parent (e.g. adds an attribute/method)
- Overriding
  - Child class redefines parent method
- Implementation/reification
  - Child class provides the actual behaviour of a parent method

Wrap-up session

- Polymorphism
  - The same message can produce different behavior depending on the actual type of the receiver objects (late binding of message/method)

SOftEng
http://softeng.polito.it

License (1)

- THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.
- BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.
- **1. Definitions**
  - **"Collective Work"** means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
  - **"Derivative Work"** means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
  - **"Licensor"** means the individual or entity that offers the Work under the terms of this License.
  - **"Original Author"** means the individual or entity who created the Work.
  - **"Work"** means the copyrightable work of authorship offered under the terms of this License.
  - **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

b. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Derivative Works. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(d) and 4(e).

License (2)

- **4. Restrictions.** The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:
  a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(c), as requested.
  b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
  c. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; and to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.
  d. For the avoidance of doubt, where the Work is a musical composition:
    i. Performance Royalties Under Blanket Licenses. Licensor reserves the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.
    ii. Mechanical Rights and Statutory Royalties. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private monetary compensation.
  - **Webcasting Rights and Statutory Royalties.** For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

License (3)

- **5. Representations, Warranties and Disclaimer**
- UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.
- **6. Limitation on Liability.** EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
- **7. Termination**
  a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
  b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

  a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
  b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
  c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
  d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.