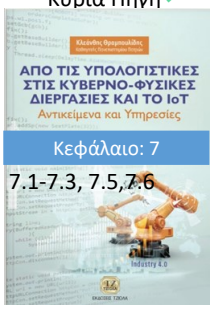
Αντικειμενοστρεφής Προγραμματισμός (Object-Oriented Programming)

(CEID_NNY106)

Κληρονομικότητα- Πολυμορφισμός (Inheritance-Polymorphism)

Κύρια Πηγή



Inheritance:
the act of inheriting property

Kleanthis Thramboulidis
Prof. of Software and System Engineering
University of Patras
https://sites.google.com/site/thramboulidiskleanthis/

Java

High-level programming language

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. Wikipedia

Designed by: James Gosling

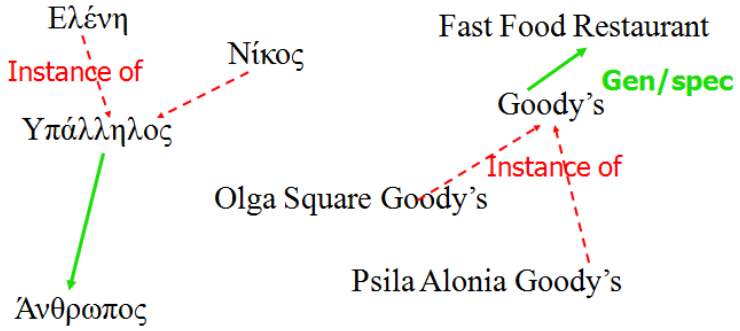
First appeared: May 23, 1995; 27 years ago

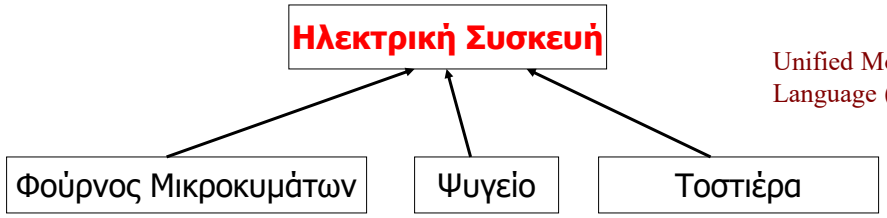
Paradigm: Multi-paradigm: generic, object-oriented (class-based), functional, imperative, reflective, concurrent

Οργάνωση Διάλεξης

- **Εισαγωγή στην Κληρονομικότητα**
  - **Βασικές έννοιες**
  - Παράδειγμα Κληρονομικότητας (WindowsApp)
  - Abstract μέθοδοι και Abstract κλάσεις
  - Shadowed variables-Overriding methods
- Πολλαπλή κληρονομικότητα (Multiple inheritance)
- Πολυμορφισμός (Polymorphism)
  - Πολυμορφισμός στην WindowsApp
- Building the inheritance tree
  - Alternatives in writing a sub class
  - Access level modifiers

Σχέση Γενίκευσης-Εξειδίκευσης





Unified Modeling Language (UML)

Η κλάση GraphicCircle

```
public class GraphicCircle {
    public double x,y;
    public double r;
    Color outline, fill;
    public double circumference(){
            return 2*3,14*r;}
    public double area() {
            return 3.14*r*r;}
    public void draw(...) { ..............}
    }
```

Η κλάση Circle

```
public class Circle {
    public double x,y;
    public double r;
    public double circumference(){
            return 2*3,14*r;}
    public double area() {
            return 3.14*r*r;}
    }
```

Ορισμός ως απόγονης κλάσης

```
public class GraphicCircle extends Circle {
    Color outline, fill;
    public void draw(...) {  .....................}
    }

GraphicCircle gc = new GraphicCircle();
double area = gc.area();
```

Circle **c** = gc;

κάθε στιγμιότυπο της υποκλάσης μπορεί να θεωρηθεί και στιγμιότυπο της πρόγονης κλάσης

Python

```
class GraphicCircle(Circle):
    pass
```


Subclass constructor - Super

```
public GraphicCircle(double x, double y, double r, Color
   outline, Color fill) {
   this.x = x; this.y = y; this.r = r;
   this.outilne = outline; this.fill =
   fill;
      }
```

- κλήση του constructor της πρόγονης κλάσης

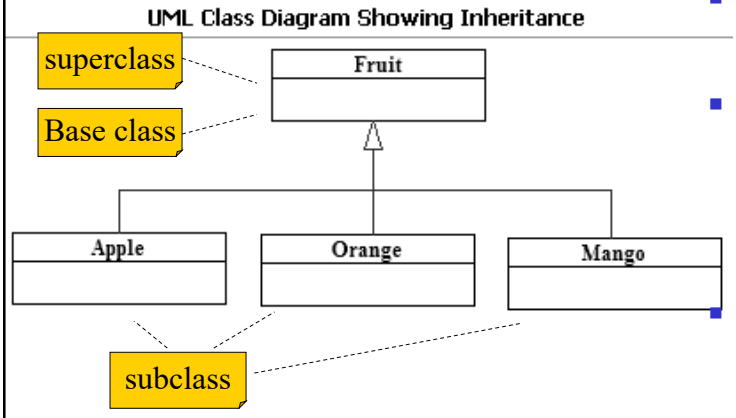
```
public GraphicCircle(double x, double y, double r, Color outline,
   Color fill) {
      super(x,y,r);
      this.outilne = outline; this.fill = fill;
      }
```

Super

- δεσμευμένη λέξη
- μέσα σε ένα constructor χρησιμοποιείται μόνο για να καλέσει τον constructor του προγόνου
- πρέπει
  - να είναι η πρώτη πρόταση του constructor
  - **να προηγείται ακόμη και από τις δηλώσεις μεταβλητών**

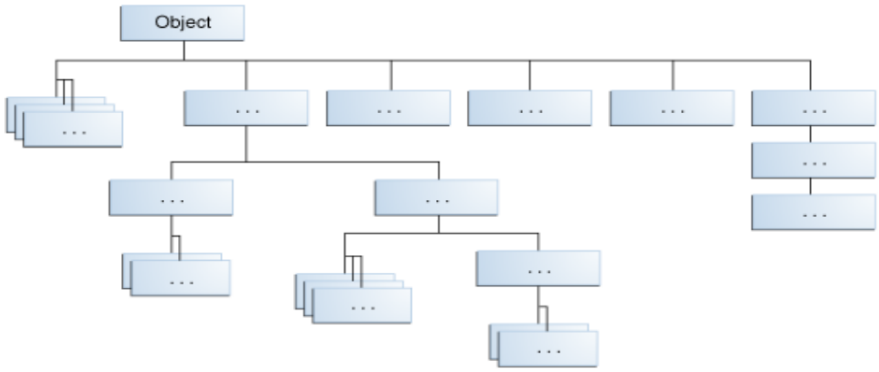
Αναπαράσταση στο Διάγραμμα Κλάσεων

inheritance tree



- **subclass**
  - A class that is derived from another class (**derived** class, **extended** class, or **child** class).
- **superclass**
  - The class from which the subclass is derived (***base*** *class* or ***parent*** *class*).
- **A subclass inherits**
  - all the *members* (fields, methods, and nested classes) from its superclass.
  - **Constructors** are not members, so they **are not inherited by subclasses**, but the constructor of the superclass can be invoked from the subclass.
- **Inheritance**
  - is a way to reuse code of existing objects, or to establish a subtype from an existing object, or both, depending upon programming language support

Java Platform Class Hierarchy



`public class Object`

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

- In the Java platform, many classes derive directly from Object, other classes derive from some of those classes, and so on, forming a hierarchy of classes.
- The **Object** class (java.lang package) defines and implements behavior common to all Java classes—including the ones that you write.

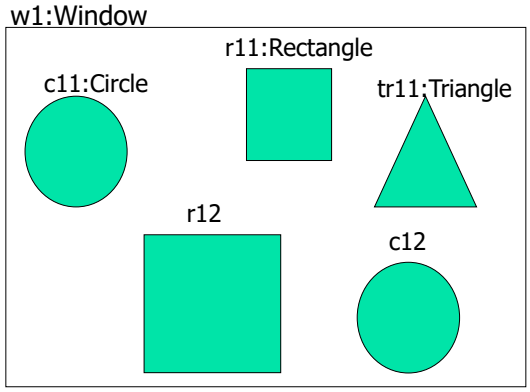
Κληρονομικότητα

- **είναι** ο μηχανισμός του περιβάλλοντος υλοποίησης που μας επιτρέπει
  - να υλοποιήσουμε την σχέση γενίκευσης/εξειδίκευσης μεταξύ των εννοιών της περιοχής του προβλήματος (**problem space**),
  - να αναπτύξουμε νέες κλάσεις της περιοχής λύσης του προβλήματος (**solution space**) οι οποίες θα κληρονομούν χαρακτηριστικά από άλλες ήδη υπάρχουσες κλάσεις.
- **Η κληρονομικότητα επιτρέπει**
  - σε νέες κλάσεις να κληρονομούν την δομή ή/και την συμπεριφορά που ορίζεται σε μία (απλή κληρονομικότητα-**single inheritance**) ή περισσότερες (πολλαπλή κληρονομικότητα-**multiple inheritance**) άλλες κλάσεις.

Οργάνωση Διάλεξης

- **Εισαγωγή στην Κληρονομικότητα**
  - Βασικές έννοιες
  - **Παράδειγμα Κληρονομικότητας (WindowsApp)**
  - **Abstract μέθοδοι και Abstract κλάσεις**
  - **Shadowed variables-Overriding methods**
- Πολλαπλή κληρονομικότητα (Multiple inheritance)
- Πολυμορφισμός (Polymorphism)
  - Πολυμορφισμός στην WindowsApp
- Building the inheritance tree
  - Alternatives in writing a sub class
  - Access level modifiers

WindowsApp – Identify Inheritance

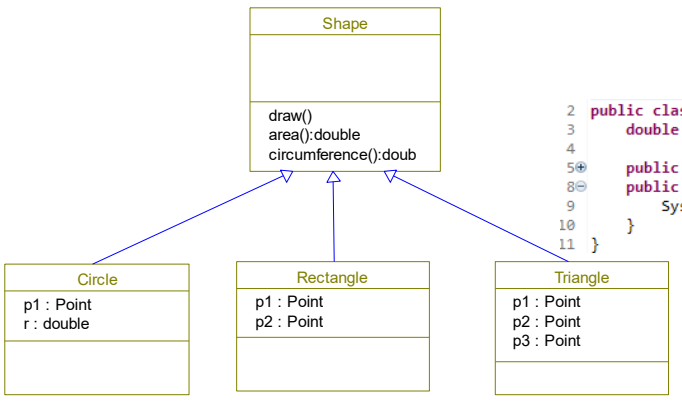



```
Circle [] circles = new Circle[100];
circles[0] = new Circle(…);
circles[1] = new Circle(…);

Rectangle [] rectangles = new Rectangle[100];
rectangles[0] = new Rectangle(…);
rectangles[1] = new Rectangle(…);
```

```
for(int i=0;i<circles.length && circles[i]!=null; i++)
        circles[i].draw();
```

Προβλήματα;

WindowsApp – Inheritance Tree




```
2  public abstract class Shape {
3      String name;
4
5      public Shape(String s) {
8      abstract void draw();
9  }
```

```
2  public class Circle extends Shape{
3      double x,y,r;
4
5      public Circle (String st) {
8      public void draw (){
9          System.out.println(this.getClass().getName()+" draw() "+this.name);
10     }
11 }
```

```
Window.draw() w1
Circle draw() c11
Rectangle draw() r11
Circle draw() c12
```

Abstract Methods and Abstract Classes

- **abstract method**
  - Είναι μια μέθοδος που δηλώνεται χωρίς να δοθεί η υλοποίηση της
    abstract *void draw();*
- **abstract class**
  - Είναι η κλάση που δηλώνεται ως abstract
  - Μπορεί αλλά δεν είναι απαραίτητο να έχει abstract μεθόδους
  - Δεν επιτρέπεται η δημιουργία στιγμιότυπων της
  - Μπορεί να χρησιμοποιηθεί ως κλάση πρόγονος ορίζοντας υποκλάσεις της

```
public abstract class Shape {
        abstract void draw(); }
```


επισκιαζόμενες μεταβλητές (shadowed variables)

```
public class GraphicCircle extends Circle {
  Color outline, fill;
  float r; //resolution in dots-per-inch.
```

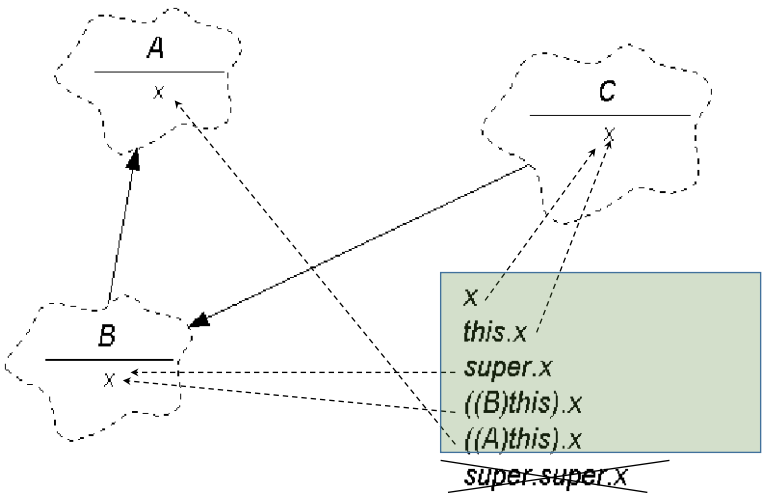
```
public class Circle {
  public double x,y;
  public double r;
  public double circumference(){
        return 2*3,14*r;}
  public double area() {
        return 3.14*r*r;}
  }
```

this.r; // αναφέρεται στη μεταβλητή resolution

super.r // αναφέρεται στην μεταβλητή ακτίνα

((Circle)this).r // cast στην κατάλληλη πρόγονη κλάση

αναφορά σε επισκιαζόμενες μεταβλητές



Overriding methods (επικαλυπτόμενες μέθοδοι)

```
class Ellipse extends Circle{
        :
        double area() {…}
```

- μέθοδος οριζόμενη σε υποκλάση με το ίδιο όνομα, παραμέτρους και επιστρεφόμενη τιμή overrides την αντίστοιχη μέθοδο της πρόγονης κλάσης

Παράδειγμα 1 – instance fields and methods

Consider also a static field

```
class A {
    int i = 1;
    int f(){return i;}
    }

class B extends A {
    // shadows variable i
    int i = 2;
    //overrides method f
    int f() {return  2*i;}
    }
```

```
public class InheritanceTest {
    public static void main(String args[]) {
        B b = new B();
        System.out.println(b.i);
        System.out.println(b.f());

        A a = (A) b;
        System.out.println(a.i);
        System.out.println(a.f());
        }
}
```

Shadowed variable

Overridden method

	Consider f() of B as **retrurn super.f();**
2	**2**
4	**1**
1	**1**
4	**1**

overriding – hiding methods

- **Instance Methods**
  - Μέθοδος στιγμιοτύπου σε υποκλάση με την ίδια υπογραφή με μέθοδο στιγμιοτύπου της υπερκλάσης (superclass) ***overrides*** την αντίστοιχη μέθοδο της υπερκλάσης.
  - Μια μέθοδος που επικαλύπτει μέθοδο πρόγονης κλάσης μπορεί να επιστρέφει τύπο που είναι subtype του τύπου που επιστρέφει η επικαλυπτόμενη μέθοδος. Αυτός ο subtype ονομάζεται *covariant return type*.
- **Static Methods**
  - Μέθοδο κλάσης με την ίδια υπογραφή με μέθοδο κλάσης της πρόγονης κλάσης (superclass) ***hides*** την αντίστοιχη μέθοδο της πρόγονης κλάσης.

Παράδειγμα 2 - methods

```
public class MethodOverrideTest {

public static void main(String[] args) {
    Cat myCat = new Cat();
    Cat.testClassMethod();
    myCat.testInstanceMethod();

    Animal myAnimal = myCat;
    Animal.testClassMethod();
    myAnimal.testInstanceMethod();

    }
}
```

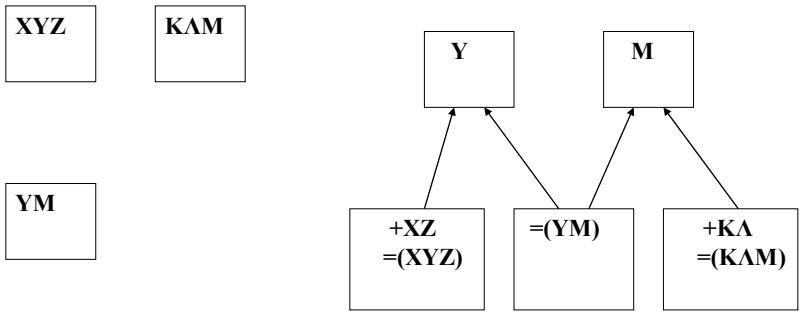
```
class Animal {
    public static void testClassMethod() {
        System.out.println("The static method in Animal");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Animal");
    }
}

class Cat extends Animal {
    public static void testClassMethod() {
        System.out.println("The static method in Cat");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Cat");
    }
}
```

```
The static method in Cat
The instance method in Cat

The static method in Animal
The instance method in Cat
```


Οργάνωση Διάλεξης

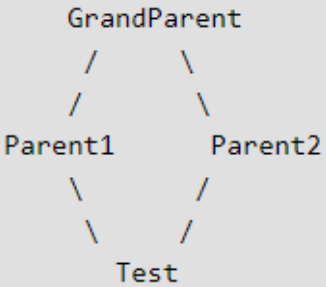
- Εισαγωγή στην Κληρονομικότητα
  - Βασικές έννοιες
  - Παράδειγμα Κληρονομικότητας (WindowsApp)
  - Abstract μέθοδοι και Abstract κλάσεις
  - Shadowed variables-Overriding methods
- **Πολλαπλή κληρονομικότητα (Multiple inheritance)**
  - **Interface – Collection interface**
- Πολυμορφισμός (Polymorphism)
  - Πολυμορφισμός στην WindowsApp
- Building the inheritance tree
  - Alternatives in writing a sub class
  - Access level modifiers

Πολλαπλή Κληρονομικότητα



- Πολλαπλή Κληρονομικότητα έχουμε όταν μια κλάση κληρονομεί χαρακτηριστικά περισσοτέρων από μία κλάσης.
- αποτελεί πηγή δημιουργίας πολλών προβλημάτων
- η χρήση της πρέπει να γίνεται με μεγάλη προσοχή και μόνο από πεπειραμένους προγραμματιστές

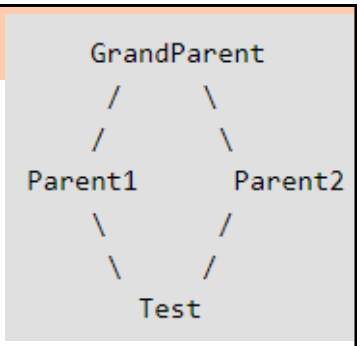
Multiple inheritance … το πρόβλημα



- Το πρόβλημα δημιουργείται όταν υπάρχουν μέθοδοι με την ίδια υπογραφή και σε δύο από της υπερκλάσεις μια υποκλάσης και στην υποκλάση.
- On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.
- Why Java doesn't support Multiple Inheritance?
  - See https://www.geeksforgeeks.org/java-and-multiple-inheritance/

The Diamond Problem

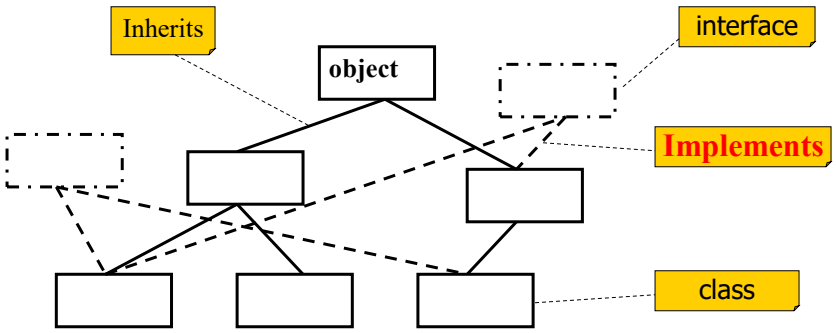
```java
// A Grand parent class in diamond
class GrandParent
{
    void fun()
    {
        System.out.println("Grandparent");
    }
}

// First Parent class
class Parent1 extends GrandParent
{
    void fun()
    {
        System.out.println("Parent1");
    }
}

// Second Parent Class
class Parent2 extends GrandParent
{
    void fun()
    {
        System.out.println("Parent2");
    }
}
```




```java
// Error : Test is inheriting from multiple
// classes
class Test extends Parent1, Parent2
{
   public static void main(String args[])
   {
       Test t = new Test();
       t.fun();
   }
}
```

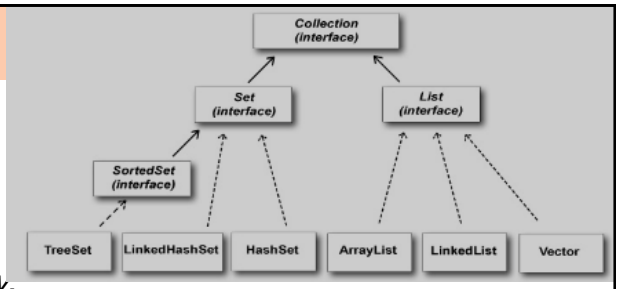
Source: https://www.geeksforgeeks.org/java-and-multiple-inheritance/

Πολλαπλή Κληρονομικότητα στην Java - Interfaces



είναι δυνατή η δήλωση μεταβλητής του συγκεκριμένου τύπου interface και η ανάθεση σε αυτή στιγμιότυπου οποιασδήποτε από τις κλάσεις που υλοποιεί το interface

Interface Collection <E>



public interface Collection<E>
extends Iterable<E>

- The root interface in the *collection hierarchy*.
- A collection represents a group of objects, known as its *elements*.
- Some collections allow **duplicate** elements and others do not. Some **are ordered** and others **unordered**.
- The JDK does not provide any *direct* implementations of this interface: it provides implementations of more specific sub interfases like **Set** and **List**.
- This interface is typically used to pass collections around and manipulate them where maximum generality is desired.

`Iterator<E>`	`iterator()` ← Returns an iterator over the elements in this collection.

Iterator

java.util

Interface Iterator<E>

is an object that can be used to loop through collections, like ArrayList.

```
ArrayList<Shape> shapes;
```

```
for(int i=0; i<shapes.length && shapes[i]!=null; i++)
    shapes[i].draw();
```

java.util

Class ArrayList<E>

`Iterator<E>`	`iterator()` Returns an iterator over the elements in this list in proper sequence.

```
for(Iterator<Shape> i= shapes.iterator(); i.hasNext();)
    i.next().draw();
```

Interface Iterator<E>

java.util

Interface Iterator<E>

All Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method and Description
default void	**forEachRemaining**(Consumer<? super E> action) Performs the given action for each remaining element until all elements have been processed or the action throws an exception.
boolean	**hasNext**() Returns true if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
default void	**remove**() Removes from the underlying collection the last element returned by this iterator (optional operation).

Iterator<E>	**iterator**() Returns an iterator over the elements in this collection.

For-Each Loop (Enhanced for statement)

designed for

- **iteration** through **Collections** and **arrays**.
- can be used to make your loops more **compact and easy to read.**

```
ArrayList<Shape> shapes = new ArrayList<Shape>();
for(Shape currentShape:shapes)
    currentShape.draw();
```

for each Shape currentShape in shapes

Οργάνωση Διάλεξης

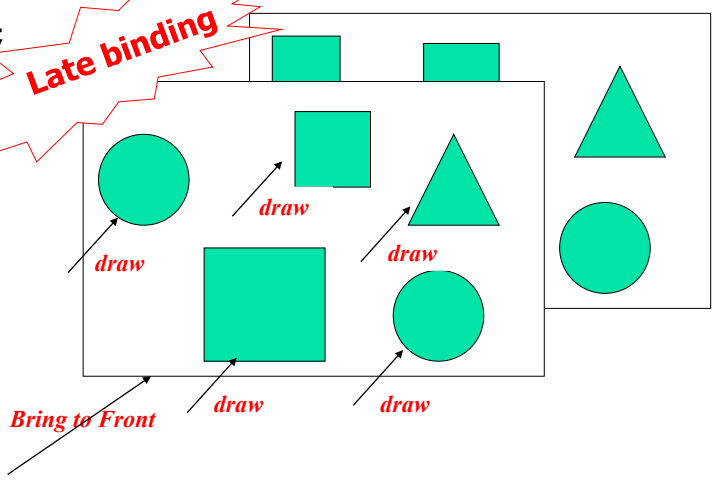
- Εισαγωγή στην Κληρονομικότητα
  - Βασικές έννοιες
  - Κληρονομικότητα στην RPN Calculator
  - Abstract μέθοδοι και Abstract κλάσεις
  - Shadowed variables-Overriding methods
- Πολλαπλή κληρονομικότητα (Multiple inheritance)
- **Πολυμορφισμός (Polymorphism)**
  - **Πολυμορφισμός στην WindowsApp**
- Building the inheritance tree
  - Alternatives in writing a sub class
  - Access level modifiers

Πολυμορφισμός – Μέσα από ένα παράδειγμα

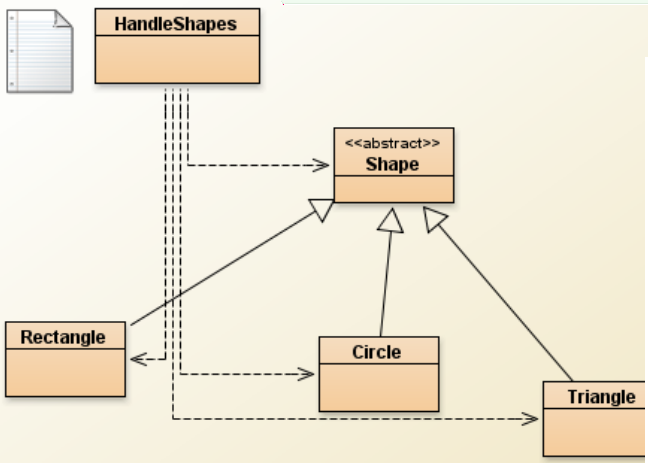
```
Shape [] shapes = new Shape[100];
shapes[0] = new Circle(…);
shapes[1] = new Rectangle(…);

for(int i=0; i<100; i++)
    shapes[i].draw();
```

Late binding

ο αποστολέας ενός μηνύματος δεν χρειάζεται να γνωρίζει την κλάση του στιγμιότυπου παραλήπτη



HandleShapes




```
public abstract class Shape
{
    public abstract void draw();
}
```

```
public class Circle extends Shape
{
    public void draw()
    {
        System.out.println("cirlce draw");
    }
}
```

```
public class HandleShapes
{
    public static void main(String[] args)
    {
        List<Shape> shapes = new LinkedList<Shape>();

        shapes.add(new Rectangle());
        shapes.add(new Circle());
        shapes.add(new Triangle());
        shapes.add(new Circle());
        shapes.add(new Rectangle());

        for (Shape currentShape: shapes)
            currentShape.draw();
    }
}
```


Early binding vs late binding

```
Circle [] circles = new Circle[100];
circles[0] = new Circle(…);
circles[1] = new Circle(…);
```

```
for(int i=0; i<100; i++)
    circles[i].draw();
```

Compile-time binding
(**early binding**)

```
Shape [] shapes = new Shape[100];
shapes[0] = new Circle(…);
shapes[1] = new Rectangle(…);
```

```
for(int i=0; i<100; i++)
    shapes[i].draw();
```

Run-time binding
(**late binding**)

Virtual Methods

- Είναι η μέθοδος της οποίας η λειτουργικότητα μπορεί να επικαλυφθεί (overridden) στις απόγονες κλάσεις από μέθοδο με την ίδια υπογραφή.
- Οι Virtual μέθοδοι επιτρέπουν σε ένα πρόγραμμα να καλεί μεθόδους οι οποίες δεν έχουν οριστεί κατά τον χρόνο μεταγλώττισης.
- **Ολές οι non-static μέθοδοι είναι by default "virtual“** (in Java).
- Δύο κατηγορίες non-static μεθόδων δεν είναι virtual:
  - final, which cannot be overridden, and
  - private methods, which are not inherited.
- C++: **virtual** keyword

  virtual void display(){

Operation vs. Method

Operation

- is an abstraction of some computation that can be performed, e.g.,

  draw() of class Shape

  operate() of class Operator

```
2  public abstract class Shape {
3      String name;
4
5      public Shape(String s) {
8      abstract void draw();
9  }
```

Method

- is one of one-or-more implementations of the operation in a *particular* class, e.g.,

  draw() of class Circle

  operate() of class Adder

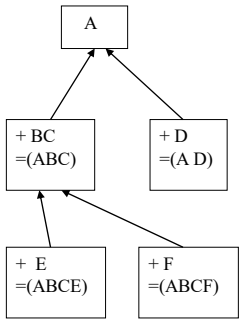
```
2  public class Circle extends Shape{
3      double x,y,r;
4
5      public Circle (String st) {
8      public void draw (){
9          System.out.println(this.getClass().getName()+" draw() "+this.name);
10     }
11 }
```

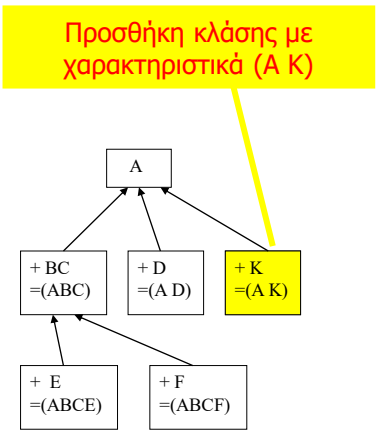
Οργάνωση Διάλεξης

- Εισαγωγή στην Κληρονομικότητα
  - Βασικές έννοιες
  - Κληρονομικότητα στην RPN Calculator
  - Abstract μέθοδοι και Abstract κλάσεις
  - Shadowed variables-Overriding methods
- Πολλαπλή κληρονομικότητα (Multiple inheritance)
- Πολυμορφισμός (Polymorphism)
  - Πολυμορφισμός στην WindowsApp
- **Building the inheritance tree**
  - Alternatives in writing a sub class
  - Access level modifiers

Προσθήκη νέας κλάσης – επιλογές

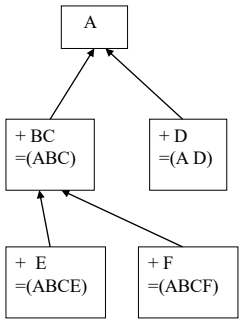
1. ορίζουμε την νέα κλάση εξ αρχής χωρίς να εκμεταλλευθούμε τα ήδη υπάρχοντα δένδρα κληρονομικότητας
2. επιλέγουμε το πλέον κατάλληλο δένδρο κληρονομικότητας και μετακινούμεθα από κάτω προς τα πάνω στην ιεραρχία μέχρι να εντοπίσουμε μια κατάλληλη πρόγονο της νέας κλάσης

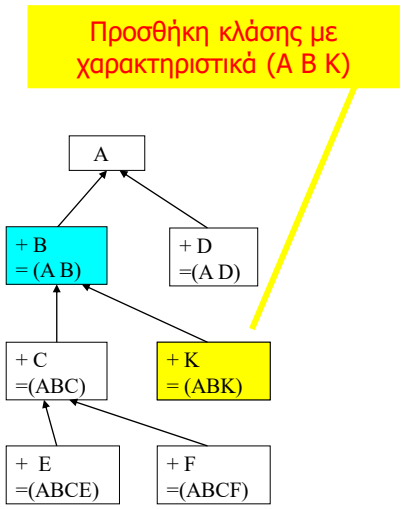




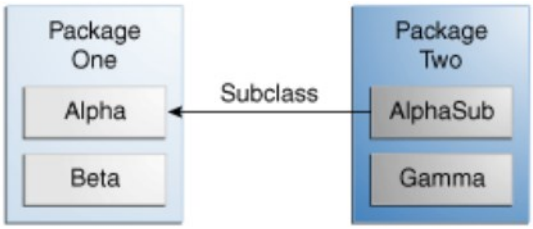
Προσθήκη νέας κλάσης – επιλογές

3. **αναδομούμε την ιεραρχία κληρονομικότητας** ώστε να δημιουργήσουμε μια κατάλληλη πρόγονο
4. Επανορίζουμε τα χαρακτηριστικά που δεν θέλουμε να κληρονομηθούν ως έχουν





Ορατότητα μελών πρόγονης κλάσης

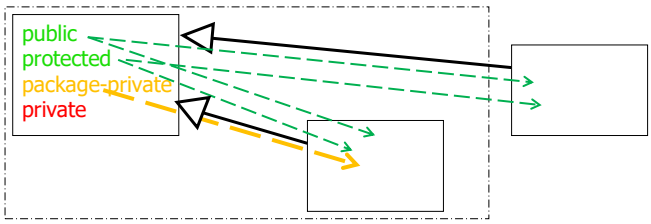


Ορατότητα των μελών της κλάσης Alpha

Modifier	Alpha	Beta	Alphasub	Gamma
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Επιλογές στο ορισμό απόγονης κλάσης 1

- Μια υποκλάση κληρονομεί όλα τα ***public*** and ***protected*** μέλη της πρόγονης κλάσης της.
- Εάν δε είναι στο ίδιο πακέτο με την πρόγονη κληρονομεί επιπλέον και τα ***package-private*** μέλη της πρόγονης κλάσης της.



Επιλογές στο ορισμό απόγονης κλάσης 2

- Στην απόγονο κλάση τα χαρακτηριστικά που κληρονομούνται μπορούν να:
  - χρησιμοποιηθούν ως έχουν (μέθοδοι πρόγονης κλάσης μπορούν να χρησιμοποιηθούν ως έχουν),
  - να αντικατασταθούν (instance μέθοδος σε υποκλάση με ίδιο όνομα με μέθοδο της πρόγονης κλάσης την υπερκαλύπτει (**overrides**)).
  - να επισκιασθούν (instance data member σε υποκλάση με ίδιο όνομα με αυτό πρόγονης κλάσης το επισκιάζει (**shadows**))
  - να συμπληρωθούν με νέα μέλη (νέες μεθόδους, νέα data members)

A nested class has access to all the private members of its enclosing class—Both fields and methods. Therefore, **a public or protected nested class inherited by a subclass has indirect access to all of the private members of the superclass**.