ΠΟΛΥΜΟΡΦΙΣΜΟΣ

ΠΟΛΥΜΟΡΦΙΣΜΟΣ

- Από τις βασικότερες έννοιες των αντικειμενοστρεφών γλωσσών προγραμματισμού, μετά την αφαιρετικότητα των δεδομένων (data abstraction) και την κληρονομικότητα (inheritance).

- Ο πολυμορφισμός σχετίζεται με την αποσύνδεση των μεθόδων από τους *τύπους.*

Τα πλεονεκτήματα της αναβάθμισης (upcasting)

ΕΠΛ233

```
import java.util.*;
class Instrument {
   public void play() {}
   static void tune(Instrument i) { i.play();}
}
class Wind extends Instrument {
   public static void main(String[] args) {
      Wind flute = new Wind();
      // Upcasting
      Instrument.tune(flute);
}}
```

- Γιατί να μην χρησιμοποιήσουμε **υπερφόρτωση** στον ορισμό της **tune()**, ώστε αυτή να δέχεται παράμετρο κλάσης **Wind;**

Χωρίς upcasting

ΕΠΛ233

```
public class Note {
  private int value;

  private Note(int val) {
      value = val;
   }

 public static final Note
    MIDDLE_C = new Note(0),
    C_SHARP = new Note(1),
    B_FLAT = new Note(2);
    }
```

Χωρίς upcasting

ΕΠΛ233

```
public class Note {
  private int value;

  private Note(int val) {
      value = val;
   }

 public static final Note
    MIDDLE_C = new Note(0),
    C_SHARP = new Note(1),
    B_FLAT = new Note(2);
    }
```

```
package polymorphism.music;

public enum Note {
  MIDDLE_C, C_SHARP, B_FLAT
}
```

Παράδειγμα (συνέχεια)

ΕΠΛ233

```
package polymorphism.music;
 class Instrument {
   public void play(Note n) {
               mypr("Inst.play()");
   }
 }
 ---------------------------------------
package polymorphism.music;
 class Wind extends Instrument {
   public void play(Note n) {
               mypr("Wind.play()");
   }
 }
```

Παράδειγμα (συνέχεια)

ΕΠΛ233

```
package polymorphism.music;

class Stringd extends Instrument {
  public void play(Note n){
     mypr("Str.play()");
  }
}

class Brass extends Instrument {
  public void play(Note n) {
    mypr("Brass.play()");
  }
}
```

Παράδειγμα (συνέχεια)

ΕΠΛ233

```
public class Music2 {
  public static void tune(Wind i) {
                i.play(Note.MIDDLE_C);
  }
  public static void tune(Stringd i){
                i.play(Note.MIDDLE_C);
  }
  public static void tune(Brass i){
                i.play(Note.MIDDLE_C);
  }
 public static void main(String[] args){
    Wind flute = new Wind();
    Stringed violin = new Stringed();
    Brass frenchHorn = new Brass();
    tune(flute); // No upcasting
    tune(violin);
    tune(frenchHorn);
 }}
```

Συνέπειες απουσίας upcasting

ΕΠΛ233

- Χρειάζεται να γράψουμε ειδικές μεθόδους **tune** για κάθε νέα κλάση που κληρονομεί από την **Instrument.** Δηλ. πρέπει να γράψουμε *περισσότερο* κώδικα.
- Επίσης, αν θελήσουμε να προσθέσουμε καινούριες μεθόδους (σαν την **tune**) ή καινούριες κλάσεις (τύπου **Instrument**), υπάρχει πολλή δουλειά που πρέπει να κάνουμε.
  - Ο μεταφραστής δεν θα μας ενημερώσει αν δεν κάνουμε υπερφόρτωση *όλων* των μεθόδων της Instrument.
- Το **upcasting** και ο πολυμορφισμός μας επιτρέπουν να ορίσουμε **μια μέθοδο,** στην **κλάση-βάσης**, και αυτή τη μέθοδο να την χρησιμοποιήσουμε και σε αντικείμενα κλάσεων-κληρονόμων.

Βασικό Ερώτημα

ΕΠΛ233

```
static void tune(Instrument i)      {
        i.play(Note.MIDDLE_C);
}
```

- Όπως είπαμε, στην **tune** μπορούμε να "περάσουμε" χειριστήρια αντικειμένων που κληρονομούν από την κλάση **Instrument.**
- Εφόσον μια κλάση-κληρονόμος καθορίζει τη "δική" της μέθοδο **play**, η κλήση της **play** μέσα από την **tune** θα γίνει σύμφωνα με αυτή την "ειδική" μέθοδο, όταν στην **tune** περάσουμε χειριστήριο της κλάσης-κληρονόμου.
- Πώς μπορεί να γνωρίζει ο μεταγλωττιστής ότι η παράμετρος που περνάει σε κάποιο σημείο προς την **tune** είναι κλάσης **Wind** και όχι **Brass** ή **Stringed,** ώστε να καλέσει την κατάλληλη **tune;**

Όψιμη (Καθυστερημένη) Πρόσδεση

- Ο μεταφραστής δεν μπορεί να γνωρίζει στο χρόνο-μετάφρασης ποιές παράμετροι θα περαστούν στην **tune** και πότε...
- Η λύση που υιοθετείται από τις Α/Σ ΓΠ λέγεται **όψιμη πρόσδεση (late binding)** ή **δυναμική πρόσδεση** ή **πρόσδεση στον χρόνο εκτέλεσης** (run-time binding).
- Η όψιμη πρόσδεση στηρίζεται σε έναν μηχανισμό, ο οποίος μπορεί να καθορίσει τον τύπο ενός αντικειμένου την στιγμή της εκτέλεσης, και να καλέσει την κατάλληλη μέθοδο.
- Στην Java η όψιμη πρόσδεση εφαρμόζεται για όλες τις μεθόδους, εκτός των **static** και των **final.**

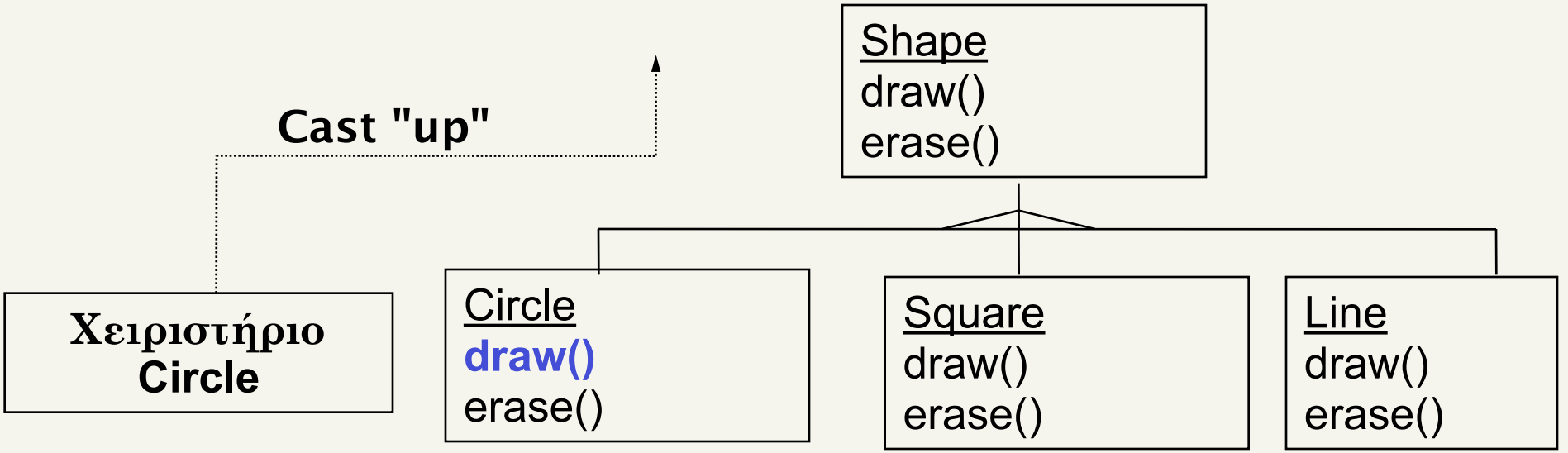
Πολυμορφική συμπεριφορά

ΕΠΛ233

- Δεδομένου ότι η Java υποστηρίζει όψιμη πρόσδεση μεθόδων, μπορείτε να αναπτύσσετε μεθόδους για μια βασική κλάση (base class) γνωρίζοντας ότι όλοι οι κληρονόμοι της θα μπορούν να χρησιμοποιούν σωστά τον ίδιο κώδικα.
- Με άλλα λόγια, μπορείτε να στέλνετε ένα μήνυμα σε ένα αντικείμενο, και να αφήνετε το αντικείμενο ν' αποφασίσει πώς θα «εκτελέσει» το μήνυμα.

Πολυμορφικές κλήσεις

ΕΠΛ233



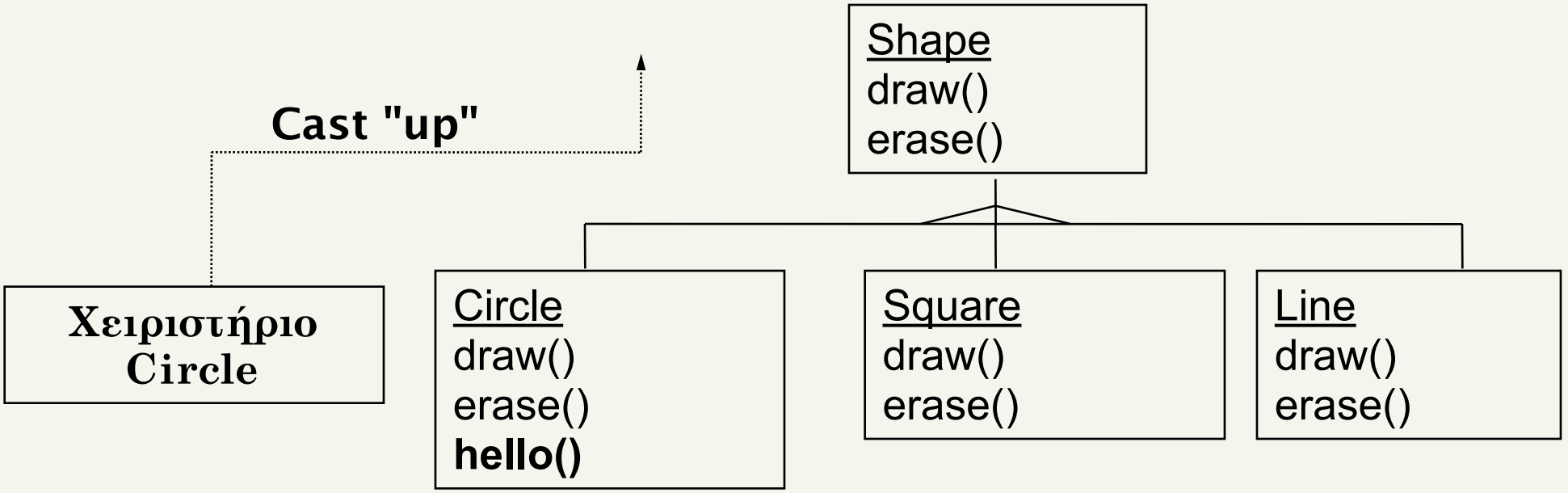
Shape s = new Circle(); // upcasting

s.draw(); // polymorphic call results to calling Circle.draw

- Η κλήση της μεθόδου **s.draw()**, η οποία έχει οριστεί στην κλάση-κληροδότη, έχει ως αποτέλεσμα την κλήση της **Circle.draw(),** λόγω όψιμης πρόσδεσης.

Πολυμορφικές κλήσεις

ΕΠΛ233




```
Shape s = new Circle(); // upcasting
s.hello();
```

Προκαλεί σφάλμα μετάφρασης

Πολυμορφικές κλήσεις

ΕΠΛ233

```
public class Shapes {
  public static Shape randShape() {
          switch((int)(Math.random() * 3)) {
          default:
          case 0: return new Circle();    // upcasting
          case 1: return new Square();    // upcasting
          case 2: return new Triangle();  // upcasting
          }
  }
  public static void main(String[] args) {
    Shape[] s = new Shape[9];
          for (int i = 0; i < s.length; i++)
                s[i] = randShape();
          for (int i = 0; i < s.length; i++)
                s[i].draw(); // polymorphic calls
} }
```

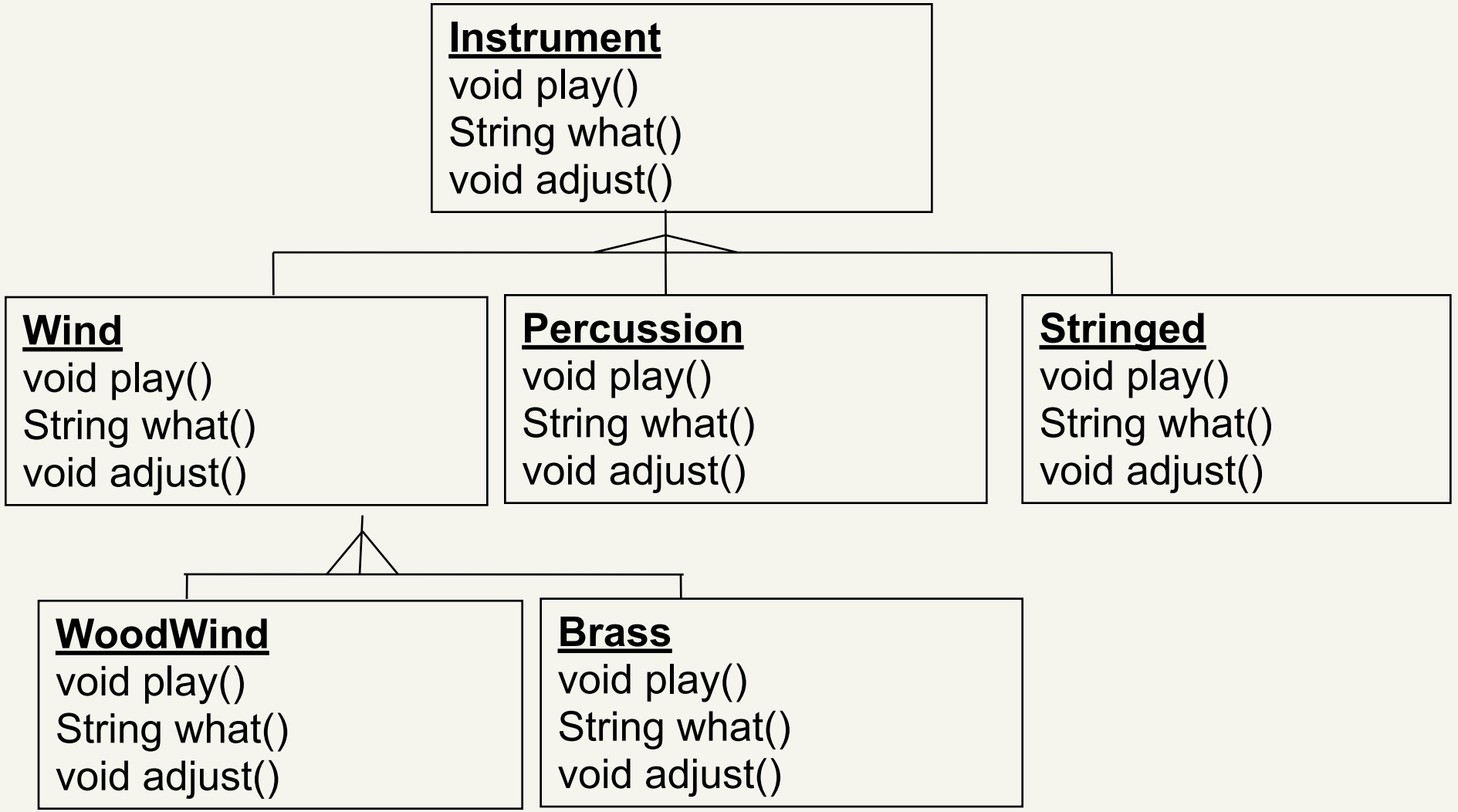
Πολυμορφισμός κ. επεκτασιμότητα

- Η κληρονομικότητα και ο πολυμορφισμός απλοποιούν την πρόσθεση νέων κλάσεων, που κληρονομούν από την κοινή κλάση-κληροδότη.

- Οι μέθοδοι που διατίθενται στην διαπροσωπεία της κλάσης-κληροδότη δεν χρειάζεται ν' αλλάξουν για την εξυπηρέτηση των νέων κλάσεων.

Πολυμορφισμός κ. επεκτασιμότητα

ΕΠΛ233



Πολυμορφισμός κ. επεκτασιμότητα

- Π.χ. οι κλάσεις **Brass** και **WoodWind** μπορούν να χρησιμοποιήσουν την **tune,** η οποία έχει οριστεί πριν καν την δημιουργία τους.
- Ακόμη κι αν η **tune()** βρίσκεται σε διαφορετικό αρχείο και προσθέσουμε νέες μεθόδους στην διαπροσωπεία τής **Instrument**, η **tune()** θα δουλέψει σωστά χωρίς επαναμετάφραση.

Επεκτασιμότητα (παράδειγμα)

ΕΠΛ233

```
import java.util.*;
class Instrument {
  public void play() {   }
  public String what() {}
  public void adjust() {}
}
class Wind extends Instrument {
  public void play() { }
  public String what() {}
  public void adjust() {}
}
class Percussion extends Instrument {
  public void play() { }
  public String what() { }
  public void adjust() { }}
```

Επεκτασιμότητα (παράδειγμα)

ΕΠΛ233

```
class Stringed extends Instrument {
  public void play() { }
  public String what() { }
  public void adjust() { }
}
class Brass extends Wind {
  public void play() {}
  public void adjust() {}
}

class Woodwind extends Wind {
  public void play() {}
  public String what() {}
}
```

Επεκτασιμότητα (παράδειγμα)

ΕΠΛ233

```
public class Music3 {
  static void tune(Instrument i) {
    // ...
    i.play();
  }
  static void tuneAll(Instrument[] e)
    {    for(int i = 0; i < e.length; i++)    tune(e[i]);  }
  public static void main(String[] args){
     Instrument[] orchestra = new Instrument[5];
     int i = 0;
     orchestra[i++] = new Wind();
     orchestra[i++] = new Percussion();
     orchestra[i++] = new Stringed();
     orchestra[i++] = new Brass();
    orchestra[i++] = new Woodwind();
     tuneAll(orchestra);
  }
```

Doesn't care about type, so new types added to the system still work right

Upcasting during addition to the array

Παρατηρήσεις για κληρονομικότητα

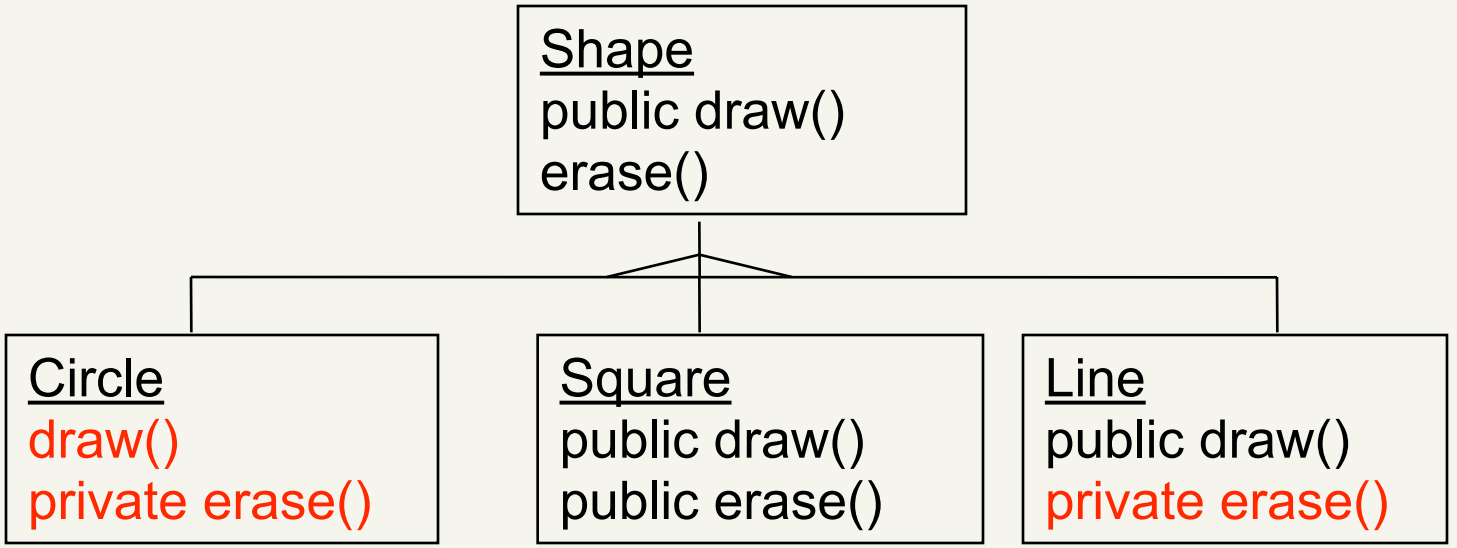
- Μια υποκλάση κληρονομεί όλα τα μέλη (πεδία δεδομένων και μεθόδους) της υπερκλάσης της.
- Ωστόσο, η υποκλάση μπορεί να μην έχει πρόσβαση σε κάποιο κληρονομημένο μέλος, όπως π.χ. στα ιδιωτικά (private) μέλη που κληρονομούνται από την υπερκλάση.
- Σημείωση: οι κατασκευαστές δεν θεωρούνται μέλη των κλάσεων και γι' αυτό δεν κληρονομούνται.

Κληρονομικότητα και προσδιοριστές πρόσβασης

ΕΠΛ233

- Η κλάση-κληρονόμος μπορεί να υπερσκελίσει (override) την λειτουργικότητα μιας μεθόδου της υπερκλάσης της δηλώνοντας μια μέθοδο με την ίδια υπογραφή και τον ίδιο τύπο επιστροφής με αυτόν της μεθόδου της υπερκλάσης.
- Η δυνατότητα της υπερσκέλισης μας επιτρέπει να κληρονομούμε συμπεριφορές από τις υπερκλάσεις μας και να τροποποιούμε αυτές τις συμπεριφορές μόνο όταν το κρίνουμε αναγκαίο.
- Ο προσδιοριστής πρόσβασης της υπερσκελίζουσας μεθόδου μιας υποκλάσης, μπορεί να δίνει τα ίδια ή περισσότερα δικαιώματα πρόσβασης με αυτά της μεθόδου της υπερκλάσης, όχι όμως λιγότερα.
  - Π.χ: μια friendly μέθοδος της υπερκλάσης μπορεί να γίνει public αλλά όχι private

Κληρονομικότητα και προσδιοριστές πρόσβασης



- Αν επιχειρήσουμε να αλλάξουμε (*περιορίσουμε*) τις δικαιοδοσίες πρόσβασης σε μεθόδους της Shape:
  - θα πάρουμε μύνημα σφάλματος: δεν μπορούμε να περιορίσουμε την "ορατότητα" των μελών της διαπροσωπείας της Shape!

Πολυμορφισμός και Πεδία Δεδομένων


```
class Super {
     public int field = 0;
     public int getField() { return field; }
}
class Sub extends Super {
  public int field = 1;
  public int getField() { return field; }
  public int getSuperField() { return super.field; }
}
public class Field Access {
     public static void main(String[] args) {
       Super sup = new Sub(); // upcasting
       System.out.println(sup.field + " " + sup.getField());
       Sub sub = new Sub();
       System.out.println(sub.field + " " + sub.getField() + " " + sub.getSuperField());
     }
 }
```

- Η απευθείας πρόσβαση σε ένα πεδίο δεδομένων επιλύεται στον χρόνο

Επισκίαση (hiding) vs. Υπερσκέλιση (overriding)

ΕΠΛ233

- Αν μια υποκλάση ορίζει μια στατική μέθοδο με την ίδια υπογραφή με μια επίσης στατική μέθοδο τής υπερκλάσης της, η στατική μέθοδος της υποκλάσης επισκιάζει (*hides)* την ομώνυμη μέθοδο της υπερκλάσης.
- Η διαφοροποίηση ανάμεσα στην επισκίαση (hiding) και την υπερσκέλιση (overriding) είναι λεπτή και έχει σημαντικές προεκτάσεις.

Επισκίαση vs. Υπερσκέλιση (παράδειγμα)


```
public class Animal {

  public static void hide() {
      System.out.println(“hide() in Animal");
  }

  public void override() {
      System.out.println("override() in Animal");
  }
}
```

Επισκίαση vs. Υπερσκέλιση (παράδειγμα)


```
public class Cat extends Animal {
    public static void hide() {
        System.out.println("hide() in Cat.");
    }
    public void override() {
        System.out.format(“override() in Cat.");
    }
    public static void main(String[] args) {
         Animal myAnimal = new Cat();
        myAnimal.hide();   // BAD STYLE
        Animal.hide();          // Better!
        myAnimal.override();
    }}
```

Επισκίαση vs. Υπερσκέλιση

ΕΠΛ233

- Στην περίπτωση κλήσης μιας στατικής μεθόδου πάνω σε χειριστήριο **x** κάποιας κλάσης **A**, το σύστημα εκτέλεσης θα καλέσει τη στατική μέθοδο που δηλώνεται ως μέλος της κλασης **A**, *ασχέτως του τύπου του αντικειμένου στο οποίο παραπέμπει το χειριστήριο*.
- Στην περίπτωση κλήσης μιας μη-στατικής μεθόδου πάνω σε χειριστήριο **x** κάποιας κλάσης **A**, το σύστημα εκτέλεσης θα καλέσει τη μέθοδο που ανήκει στην κλάση του αντικειμένου στο οποίο παραπέμπει το x.
- Μια μη στατική μέθοδος δεν επιτρέπεται να *υπερσκελίσει* μια στατική μέθοδο.
- Μια στατική μέθοδος δεν επιτρέπεται να *επισκιάσει* μια μη στατική μέθοδο.

Defining a Method with the Same Signature as a Superclass's Method

ΕΠΛ233

	Superclass Instance Method	Superclass Static Method
Instance Method	Overrides (return type must be a subtype of the return type of the superclass's method) - υπερσκελίζει	Compile-time error
Static Method	Generates a compile-time error	Hides - Επισκιάζει

Κατασκευαστές και Πολυμορφισμός

- Ο κατασκευαστής ενός κληροδότη καλείται πάντοτε μέσα από τον κατασκευαστή του κληρονόμου του.
- Έτσι, δημιουργείται μια αλυσίδα κλήσεων προς τα ανώτερα στρώματα της ιεραρχίας κληρονομικότητας.
- Η αλυσίδα αυτή των κλήσεων διασφαλίζεται από τον μεταγλωττιστή, ώστε να εξασφαλιστεί η σωστή κατασκευή αντικειμένων κλάσεων-κληρονόμων.
- Η σειρά των κλήσεων στην περίπτωση κατασκευής ενός αντικειμένου κάποιας κλάσης κληρονόμου Α, είναι η ακόλουθη:
  - Κλήση του κατασκευαστή της υπερκλάσης της Α. Αναδρομική επανάληψη της κλήσης ώστε να κατασκευαστεί πρώτα το υποαντικείμενο της ρίζας της ιεραρχίας, ακολουθούμενο από το υποαντικείμενο του επόμενου κληρονόμου, κοκ.
  - Εκτέλεση των αρχικοποιητών (initializers) της Α, με βάση την σειρά δήλωσης των αντίστοιχων πεδίων της κλάσης.
  - Κλήση του «σώματος» του κατασκευαστή Α().

Παράδειγμα

ΕΠΛ233

```
class Meal { Meal()   {  System.out.println("Meal()"); }}
class Bread { Bread()   { System.out.println("Bread()"); }}
class Cheese { Cheese() { System.out.println("Cheese()");}}
class Lettuce {Lettuce()  { System.out.println("Lettuce()");}}
class Lunch extends Meal {  Lunch() { System.out.println("Lunch()");}}
class PortableLunch extends Lunch {
    PortableLunch() {System.out.println("PortableLunch()");}
}
class Sandwich extends PortableLunch {
  Bread b = new Bread();
  Cheese c = new Cheese();
  Lettuce l = new Lettuce();
  Sandwich() { System.out.println("Sandwich()");}
  public static void main(String[] args) {
    new Sandwich();
  }}
```

Παράδειγμα (έξοδος)

ΕΠΛ233

Meal()
Lunch()
ProtableLunch()
Bread()
Cheese()
Lettuce()
Sandwitch()

Παράδειγμα II


```
class Test extends Object {

    /** Creates new Test */
    public Test() {

        System.out.println("Test()");
    }

    void foo() {
        System.out.println("foo in Test");
    }
}
```

Παράδειγμα II

ΕΠΛ233

```
public class Test1 extends Test {
    public Test1() {  System.out.println("Test1");     }
    public Test1(Test tt) {
       System.out.println("Test1.2");
       tt.foo();
    }
    void foo() {   System.out.println("foo in Test1");    }

    public static void main(String args[]) {
       System.out.println("main");
       Test tt1 = new Test();
       Test tt2 = new Test1();
       Test tt3 = new Test1(tt1);
       Test tt4 = new Test1(tt2);}
    }
```

Πολυμορφικές κλήσεις μέσα σε κατασκευαστές


```
abstract class Glyph {

  abstract void draw();

  Glyph() {
        System.out.println("Glyph() before draw()");
        draw();
        System.out.println("Glyph() after draw()");
   }
}
```

Πολυμορφικές κλήσεις μέσα σε κατασκευαστές


```
class RoundGlyph extends Glyph {

  int radius = 1;

  RoundGlyph(int r) {
    radius = r;
    System.out.println("RoundGlyph.RoundGlyph(), radius = " + radius);
  }
  void draw() {
    System.out.println("RoundGlyph.draw(), radius = " + radius);
  }
}


public class PolyConstructors {
  public static void main(String[] args) {
    new RoundGlyph(5);
}}
```

Πολυμορφικές κλήσεις μέσα σε κατασκευαστές

ΕΠΛ233

```
Glyph() before draw()
RoundGlyph.draw(), radius = 0
Glyph() after draw()
RoundGlyph.RoundGlyph(), radius = 5
```

- Η κλήση μιας δυναμικά προσδενόμενης (αφαιρετικής) μεθόδου μέσα στον κατασκευαστή ενός αντικειμένου, χρησιμοποιεί την υπερσκελισμένη έκδοση της μεθόδου.
- Η διαδικασία αρχικοποίησης αντικειμένου έχει ως εξής:
  - Δέσμευση χώρου για το αντικείμενο και αρχικοποίηση των πεδίων του σε μηδέν.
  - Κλήση κατασκευαστών κληροδοτών. Κλήση υπερσκελισμένων μεθόδων (π.χ. της draw()).
  - Αρχικοποίηση πεδίων δεδομένων, με τη σειρά της δήλωσής τους.
  - Κλήση του σώματος του κατασκευαστή του κληρονόμου.

Πολυμορφικές κλήσεις μέσα σε κατασκευαστές

ΕΠΛ233

- Για να αποφεύγουμε αυτές τις «δυσάρεστες» συνέπειες, καλό είναι να ακολουθούμε τον ακόλουθο κανόνα για τους κατασκευαστές των αντικειμένων:
  - Κάνε όσο λιγότερα μπορείς για να φέρεις ένα αντικείμενο σε καλή κατάσταση, κι αν είναι δυνατόν μην καλείς καμιά μέθοδο στο σώμα του κατασκευαστή.
  - Οι μόνες ασφαλείς μέθοδοι για να κληθούν μέσα σε ένα κατασκευαστή, είναι οι **τελικές** μέθοδοι της κλάσης κληροδότη του.

Συμμεταβλητοί τύποι επιστροφής

ΕΠΛ233

- Covariant return types: Μια υπερσκελισμένη μέθοδος σε κάποια κλάση-κληρονόμο μπορεί να επιστρέφει τιμή τύπου, ο οποίος τύπος να είναι κληρονόμος του τύπου που επιστρέφει η μέθοδος της κλάσης κληροδότη:

```
class Grain { public String toString() { return "Grain"; } }
class Wheat extends Grain { public String toString() { return "Wheat"; }}
class Mill { Grain process() { return new Grain(); }}
class WheatMill extends Mill { Wheat process() { return new Wheat(); }}

public class CovariantReturn {
  public static void main(String[] args) {
    Mill m = new Mill();
    Grain g = m.process();
    System.out.println(g);
    m = new WheatMill();
    g = m.process();
    System.out.println(g);
  }
}
```

Σχεδιασμός με κληρονομικότητα

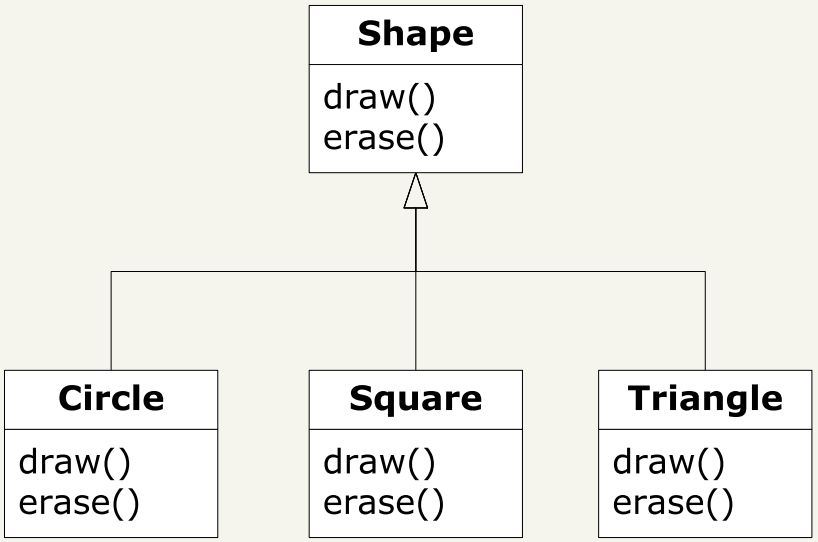
ΕΠΛ233

- Κληρονομικότητα και πολυμορφισμός δεν είναι ο μοναδικός τρόπος για την επαναχρησιμοποίηση μιας υπάρχουσας κλάσης.
- Σε αρκετές περιπτώσεις είναι προτιμότερη η χρήση της *σύνθεση*, καθώς τότε είναι δυνατή η *δυναμική* επιλογή τύπου (άρα και συμπεριφοράς). Π.χ. “State” design pattern:

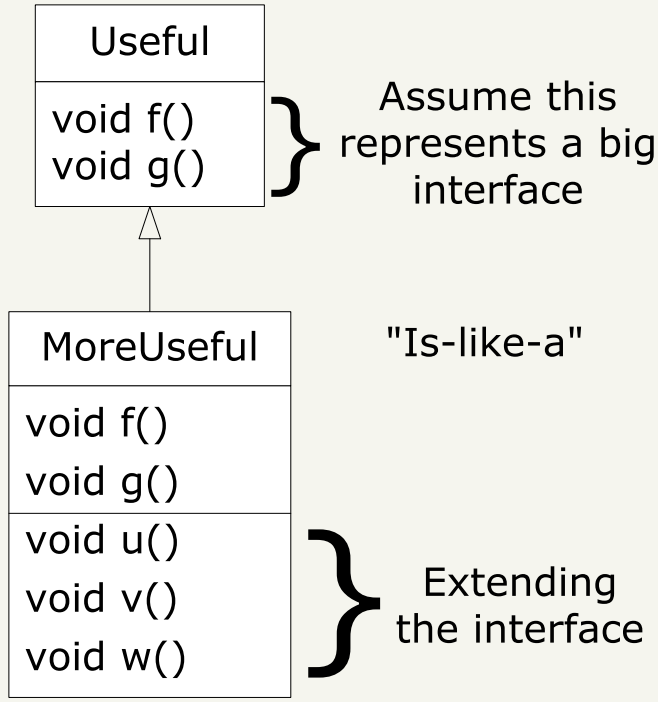
```
class Actor { public void act() {} }
class HappyActor extends Actor { public void act() { print("HappyActor");}}
class SadActor extends Actor { public void act() { print("SadActor"); }}
class Stage {
  private Actor actor = new HappyActor();
  public void change() { actor = new SadActor(); }
  public void performPlay() { actor.act(); }
}
public class Transmogrify {
  public static void main(String[] args) {
    Stage stage = new Stage();
    stage.performPlay();
    stage.change();
    stage.performPlay();
  }
```

Καθαρή κληρονομικότητα vs. Επέκταση

ΕΠΛ233



- Οι κληρονόμοι *δεν* επεκτείνουν την διαπροσωπεία του κληροδότη (*υπερσκέλιση*).



- Οι κληρονόμοι επεκτείνουν την διαπροσωπεία του κληροδότη.
- Αδύνατη η χρήση των νέων μεθόδων, μέσω Upcasting

Υποβάθμιση (downcasting)

Υποβάθμιση (downcasting)

- Όταν χρησιμοποιούμε upcasting (αναβάθμιση), ανεβαίνουμε την κλίμακα της κληρονομικότητας και χάνουμε την πληροφορία σχετικά με τον τύπο του αντικειμένου που γίνεται upcast.

Υποβάθμιση (downcasting)

- Όταν χρησιμοποιούμε upcasting (αναβάθμιση), ανεβαίνουμε την κλίμακα της κληρονομικότητας και χάνουμε την πληροφορία σχετικά με τον τύπο του αντικειμένου που γίνεται upcast.
- Η αναβάθμιση είναι ασφαλής, αφού ο κληροδότης δεν μπορεί να έχει ευρύτερη διαπροσωπεία από τους κληρονόμους του.

Υποβάθμιση (downcasting)

ΕΠΛ233

- Όταν χρησιμοποιούμε upcasting (αναβάθμιση), ανεβαίνουμε την κλίμακα της κληρονομικότητας και χάνουμε την πληροφορία σχετικά με τον τύπο του αντικειμένου που γίνεται upcast.
- Η αναβάθμιση είναι ασφαλής, αφού ο κληροδότης δεν μπορεί να έχει ευρύτερη διαπροσωπεία από τους κληρονόμους του.
- Σε περιπτώσεις που χρησιμοποιήσουμε αναβάθμιση, είναι χρήσιμο να μπορούμε να κατεβούμε την κλίμακα της κληρονομικότητας και να βρούμε τον τύπο του αντικειμένου που αναβαθμίστηκε.

Υποβάθμιση (downcasting)

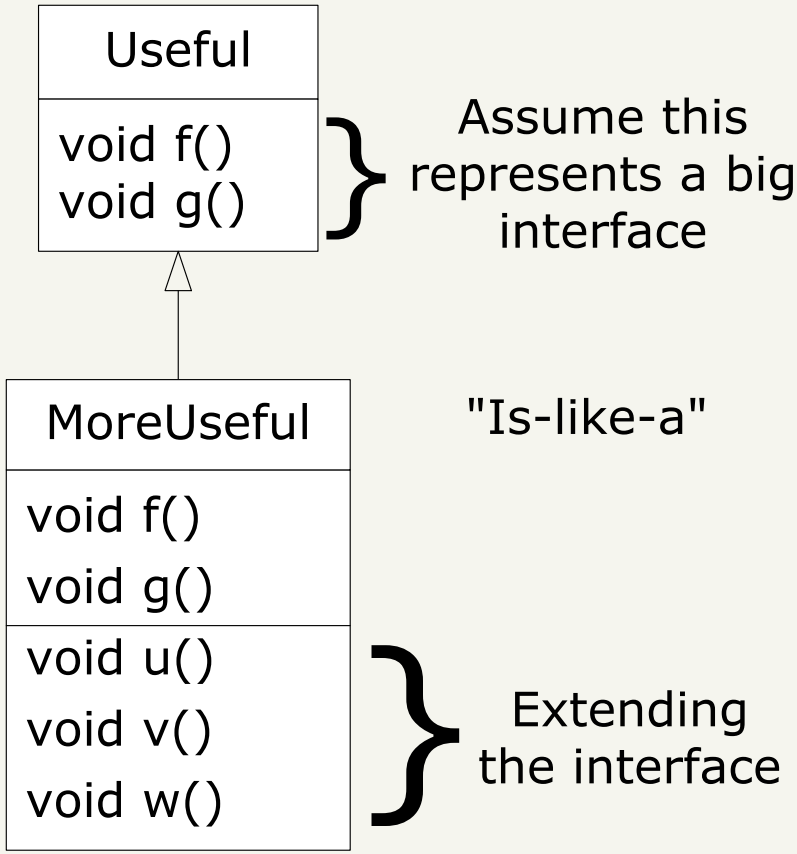
- Όταν χρησιμοποιούμε upcasting (αναβάθμιση), ανεβαίνουμε την κλίμακα της κληρονομικότητας και χάνουμε την πληροφορία σχετικά με τον τύπο του αντικειμένου που γίνεται upcast.
- Η αναβάθμιση είναι ασφαλής, αφού ο κληροδότης δεν μπορεί να έχει ευρύτερη διαπροσωπεία από τους κληρονόμους του.
- Σε περιπτώσεις που χρησιμοποιήσουμε αναβάθμιση, είναι χρήσιμο να μπορούμε να κατεβούμε την κλίμακα της κληρονομικότητας και να βρούμε τον τύπο του αντικειμένου που αναβαθμίστηκε.
- Η διαδικασία αυτή λέγεται downcasting-υποβάθμιση και εμπεριέχει τον κίνδυνο να υποβαθμίσουμε ένα χειριστήριο στο λάθος τύπο, στέλνοντας στο αντίστοιχο αντικείμενο ένα μήνυμα που δεν μπορεί να γίνει δεκτό.

Υποβάθμιση (downcasting)

ΕΠΛ233

- Όταν χρησιμοποιούμε upcasting (αναβάθμιση), ανεβαίνουμε την κλίμακα της κληρονομικότητας και χάνουμε την πληροφορία σχετικά με τον τύπο του αντικειμένου που γίνεται upcast.
- Η αναβάθμιση είναι ασφαλής, αφού ο κληροδότης δεν μπορεί να έχει ευρύτερη διαπροσωπεία από τους κληρονόμους του.
- Σε περιπτώσεις που χρησιμοποιήσουμε αναβάθμιση, είναι χρήσιμο να μπορούμε να κατεβούμε την κλίμακα της κληρονομικότητας και να βρούμε τον τύπο του αντικειμένου που αναβαθμίστηκε.
- Η διαδικασία αυτή λέγεται downcasting-υποβάθμιση και εμπεριέχει τον κίνδυνο να υποβαθμίσουμε ένα χειριστήριο στο λάθος τύπο, στέλνοντας στο αντίστοιχο αντικείμενο ένα μήνυμα που δεν μπορεί να γίνει δεκτό.
- Χρειάζεται λοιπόν κάποιος τρόπος ελέγχου αν η υποβάθμιση είναι έγκυρη. Ο έλεγχος αυτός γίνεται από την JAVA

Υποβάθμιση (downcasting)



Downcasting


```
import java.util.*;
class Useful {
  public void f() {}
  public void g() {}
}

class MoreUseful extends Useful {
  public void f() {}
  public void g() {}
  public void u() {}
  public void v() {}
  public void w() {}
}
```

Downcasting

ΕΠΛ233

```
public class RTTI {
  public static void main(String[] args) {
    Useful[] x = { new Useful(),   new MoreUseful() };
    x[0].f();
    x[1].g();
    x[1].u(); // Compile-time: method not found in Useful
    ((MoreUseful)x[1]).u(); // Downcast/RTTI
    ((MoreUseful)x[0]).u(); // Exception thrown
  }
}
```