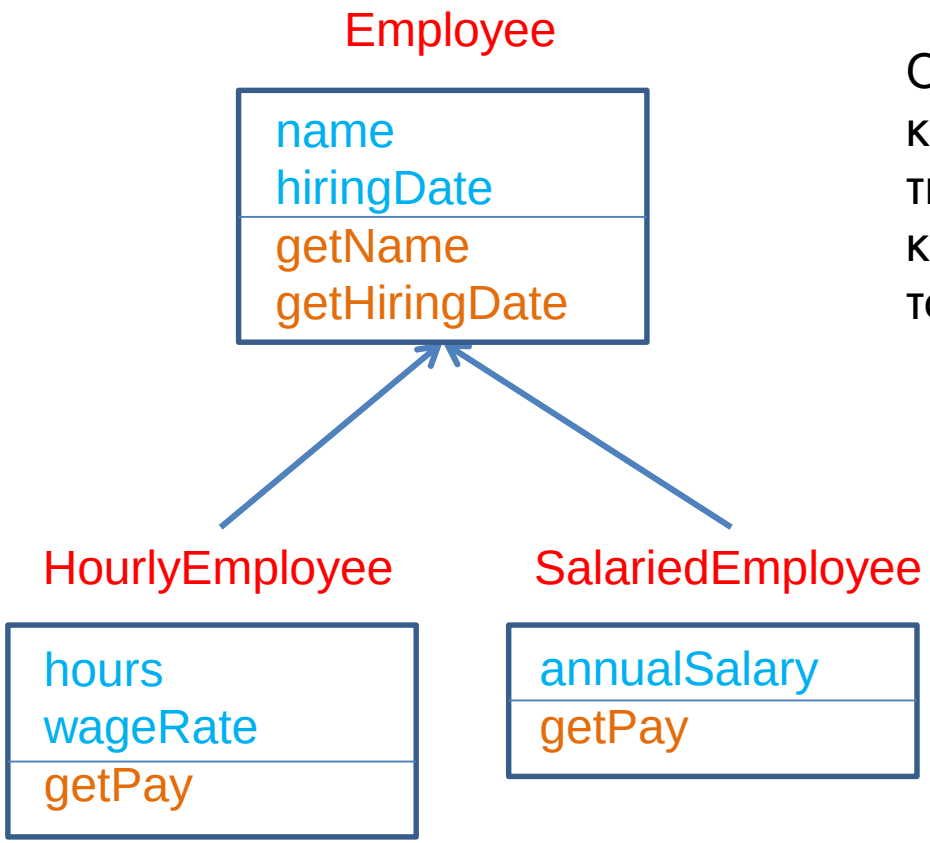
ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

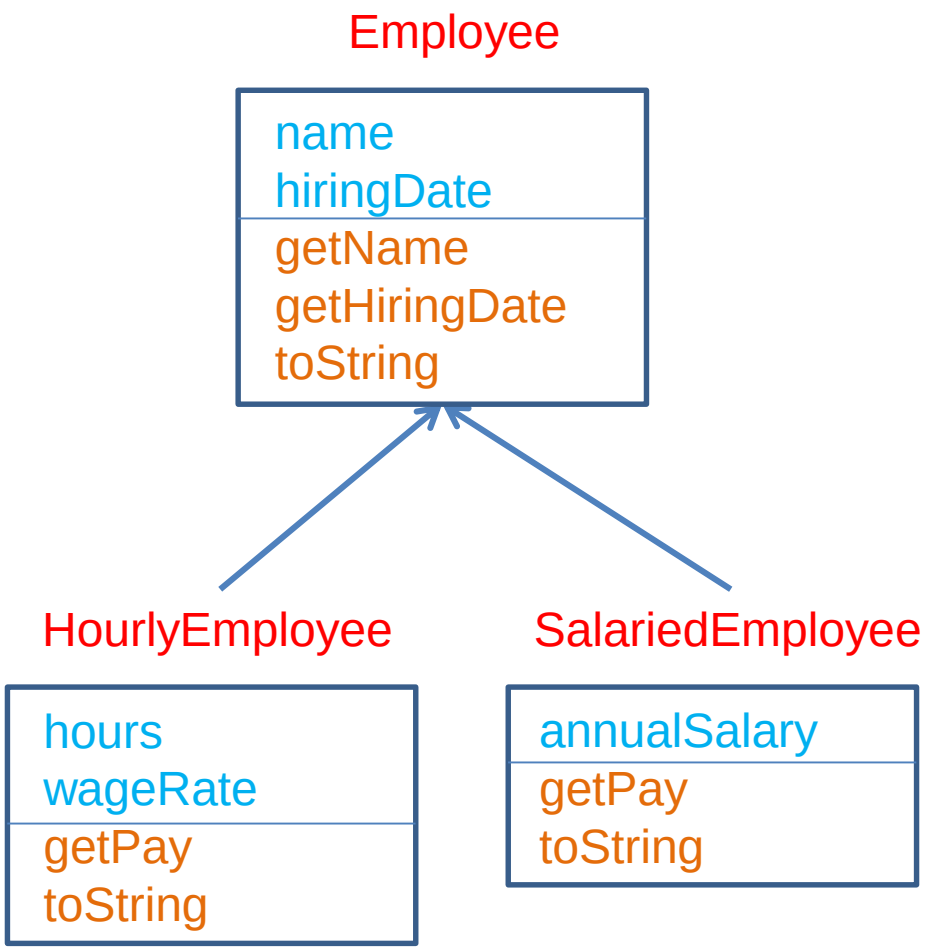
Παράδειγμα χρήσης κληρονομικότητας

Κληρονομικότητα



Οι παράγωγες κλάσεις κληρονομούν τα πεδία και τις μεθόδους της βασικής κλάσης και έχουν και δικά τους πεδία και μεθόδους

Late Binding

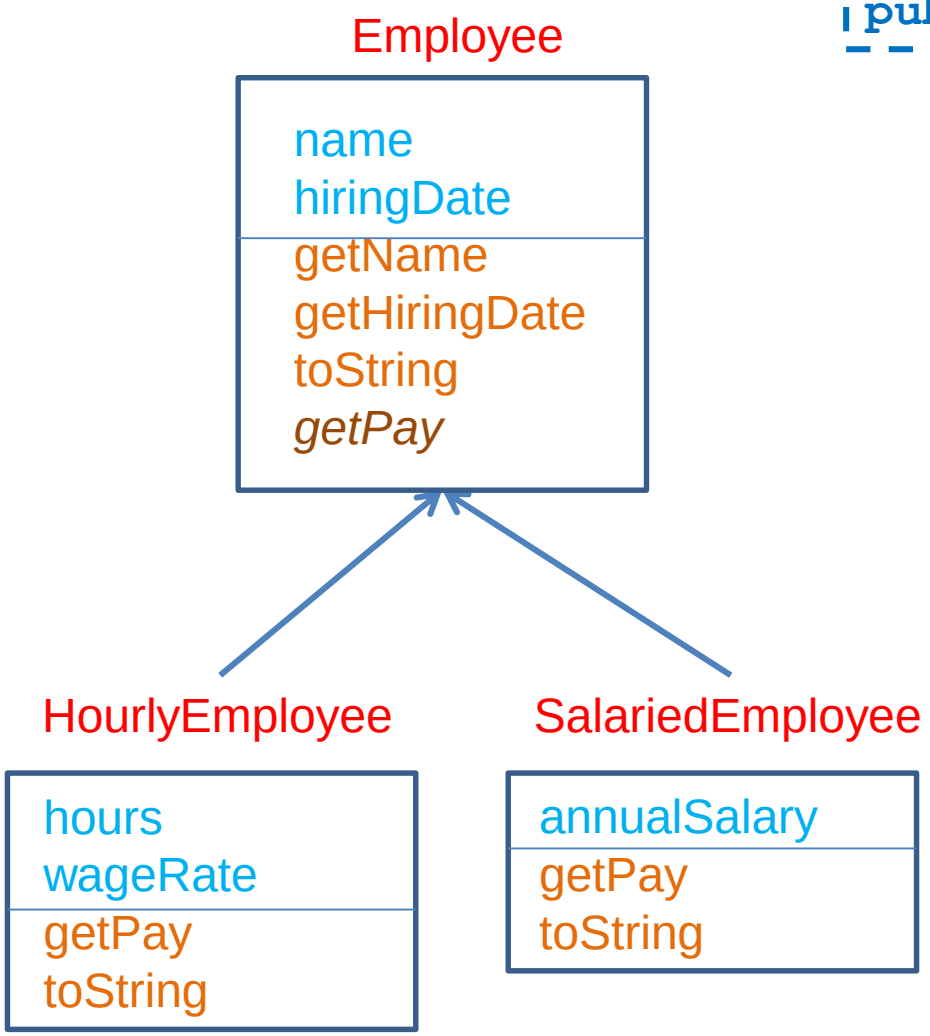



```
Employee e;
e = new HourlyEmployee();
System.out.println(e);
e = new SalariedEmployee();
System.out.println(e);
```

Late Binding:

Ο κώδικας που εκτελείται για την `toString()` εξαρτάται από την κλάση του αντικειμένου την ώρα της κλήσης (HourlyEmployee ή SalariedEmployee) και όχι την ώρα της δήλωσης (Employee)

Αφηρημένες κλάσεις




```
public abstract double getPay();
```

Μια αφηρημένη μέθοδος δηλώνεται σε μια γενική κλάση και ορίζεται σε μια πιο εξειδικευμένη κλάση

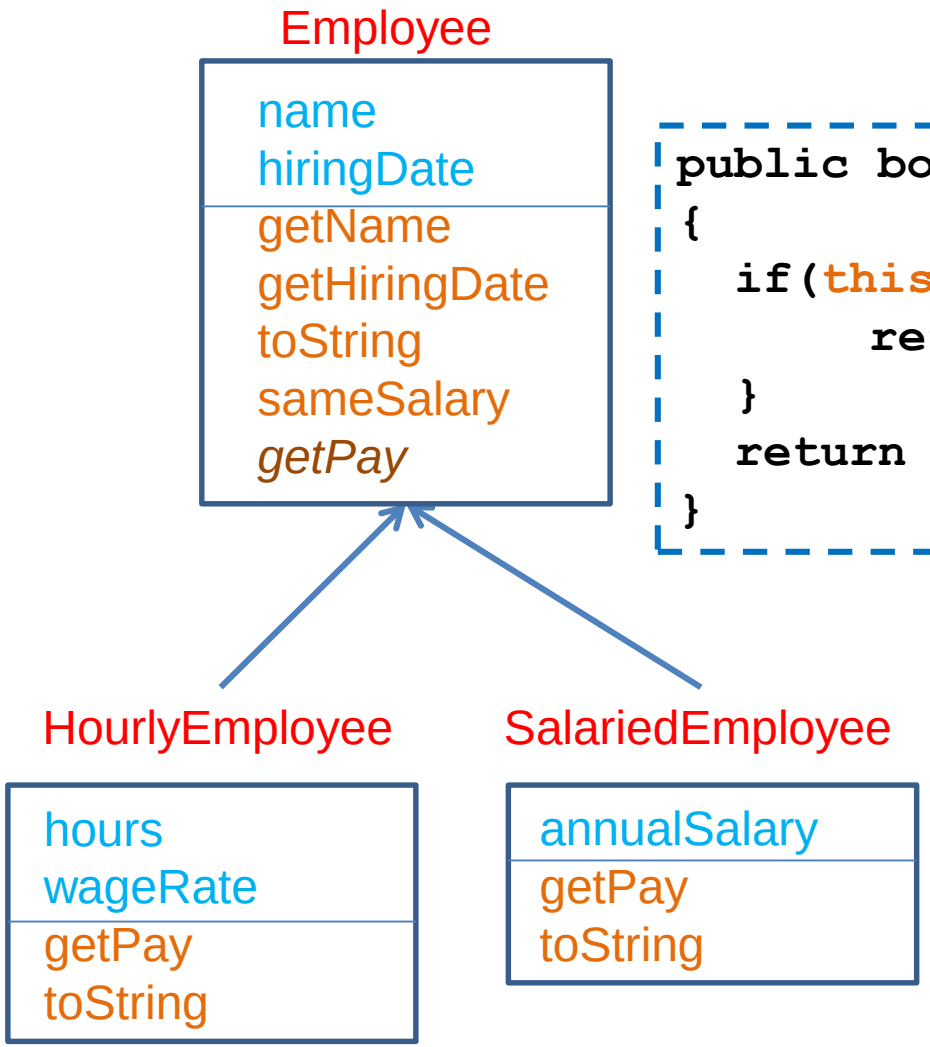
Οι κλάσεις με αφηρημένες μεθόδους είναι αφηρημένες κλάσεις.

Δεν μπορούμε να δημιουργήσουμε αντικείμενα αφηρημένων κλάσεων.

- Δηλαδή δεν μπορούμε να κάνουμε `new Employee()` εφόσον η Employee είναι αφηρημένη

Οι παράγωγες ενυπόστατες κλάσεις πρέπει να υλοποιούν τις αφηρημένες μεθόδους.

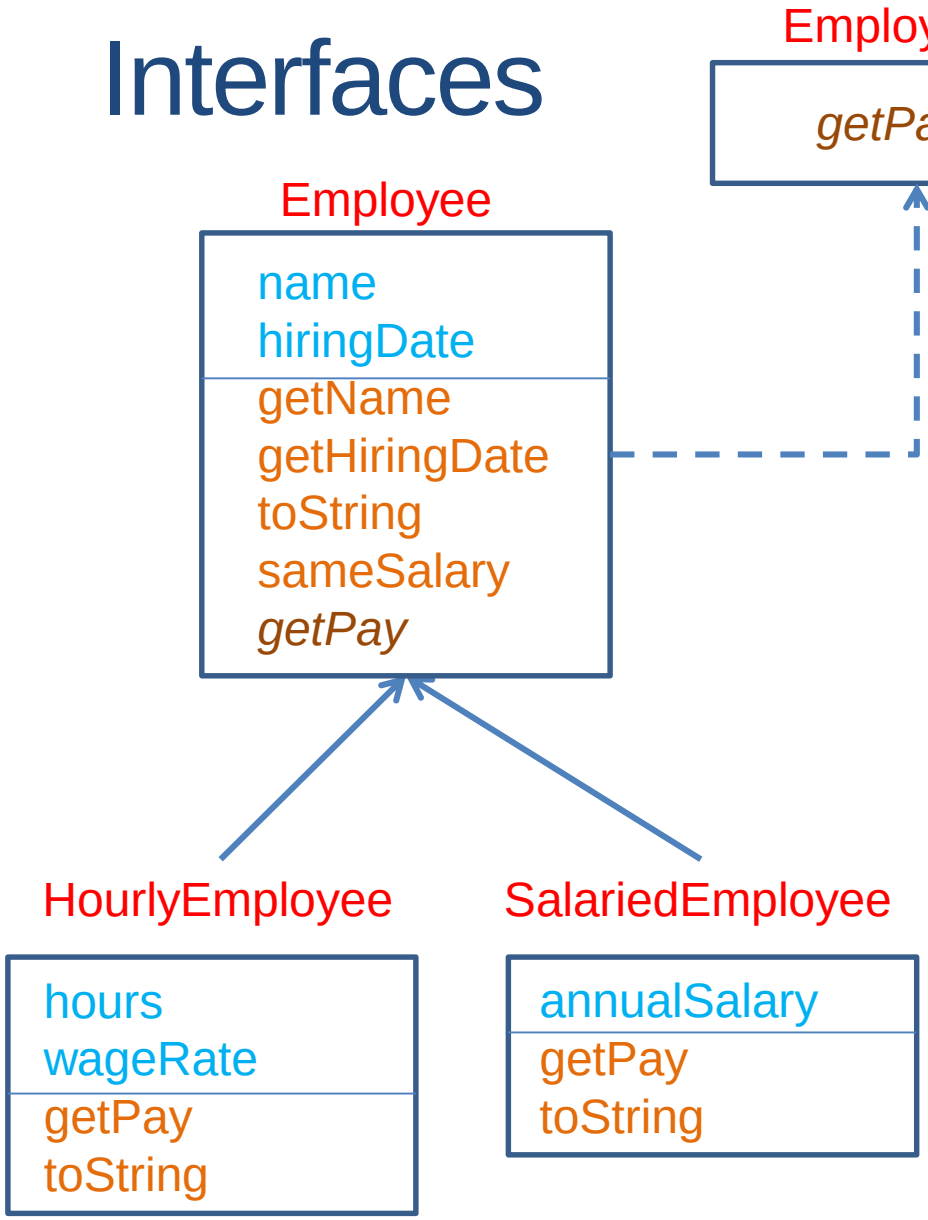
Αφηρημένες κλάσεις




```
public boolean sameSalary(Employee other)
{
   if(this.getPay() == other.getPay()){
        return true;
   }
   return false
}
```

Μια αφηρημένη μέθοδος μπορεί να χρησιμοποιηθεί μέσα στις ενυπόστατες μεθόδους της αφηρημένης κλάσης

Interfaces




```
public interface EmployeePay
{
   public double getPay();
}

public abstract Employee
        implements EmployeePay
```

Ένα interface ορίζει μια βασική λειτουργικότητα (μεθόδους).

Μία κλάση υλοποιεί το interface, δηλ. υλοποιεί τις μεθόδους του interface.

Μια κλάση μπορεί να υλοποιεί παραπάνω από ένα interfaces

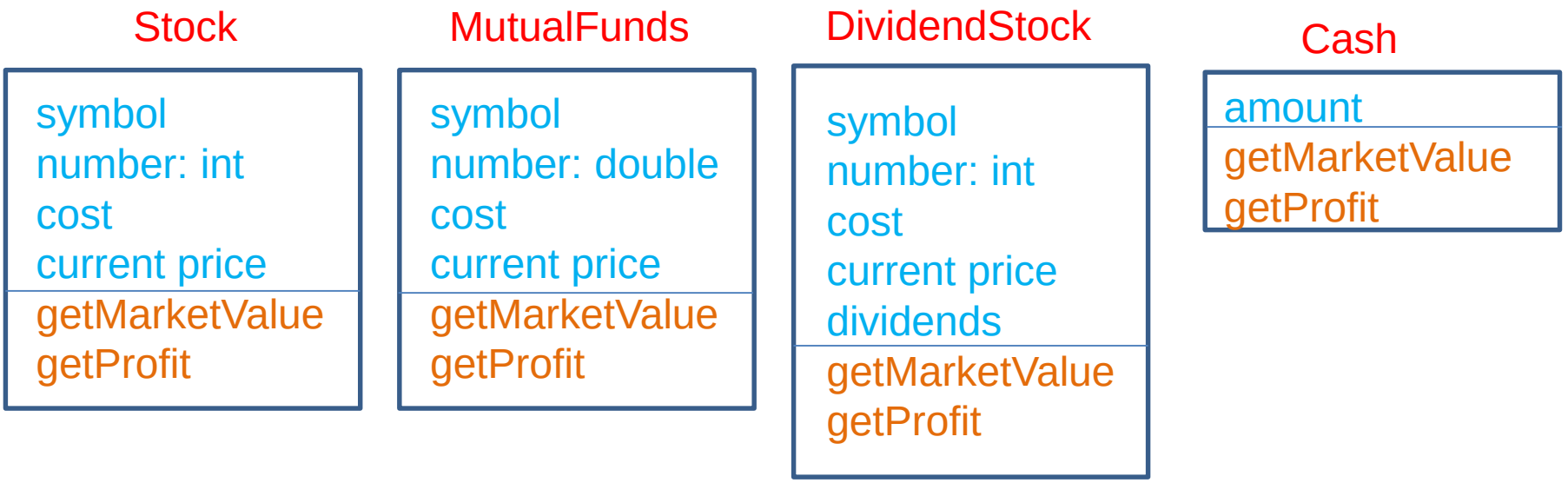
ΠΑΡΑΔΕΙΓΜΑ ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑΣ

Ένα μεγάλο παράδειγμα

- Θέλουμε να φτιάξουμε ένα πρόγραμμα που διαχειρίζεται το πορτοφόλιο (portofolio) ενός χρηματιστή. Το portofolio έχει μετοχές (stocks), μετοχές που δίνουν μέρισμα (divident stocks), αμοιβαία κεφάλαια (mutual funds), και χρήματα (cash). Για κάθε μια από αυτές τις αξίες (assets) θέλουμε να υπολογίζουμε την τωρινή της αποτίμηση (market value) και το κέρδος (profit) που μας δίνει. Μετά θέλουμε να υπολογίσουμε τη συνολική αξία του πορτοφόλιου και το συνολικό κέρδος

Λεπτομέρειες

- Cash: Δεν μεταβάλλεται η αξία του, δεν έχει κέρδος
- Stocks: Η αξία του είναι ίση με τον αριθμό των μετοχών επί την αξία της μετοχής. Το κέρδος είναι η διαφορά της τωρινής αποτίμησης με το κόστος αγοράς
- Mutual Funds: Παρόμοια με τα Stocks αλλά ο αριθμός των μετοχών που μπορούμε να έχουμε είναι πραγματικός αριθμός αντί για ακέραιος
- Dividend Stocks: Όμοια με τα Stocks αλλά στο κέρδος προσθέτουμε και τα μερίσματα

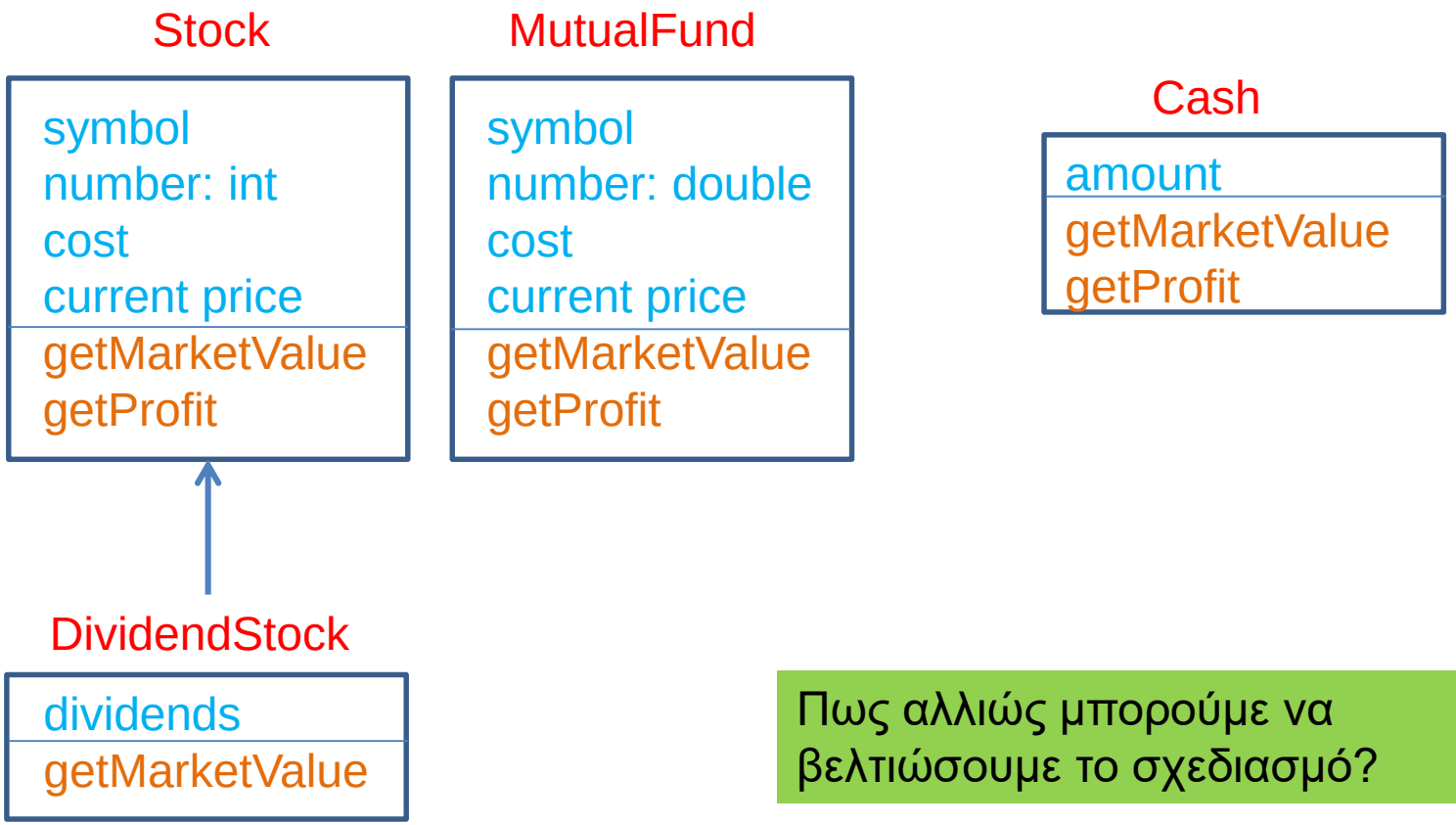


Πως μπορούμε να βελτιώσουμε το σχεδιασμό των κλάσεων?

Σχεδιασμός

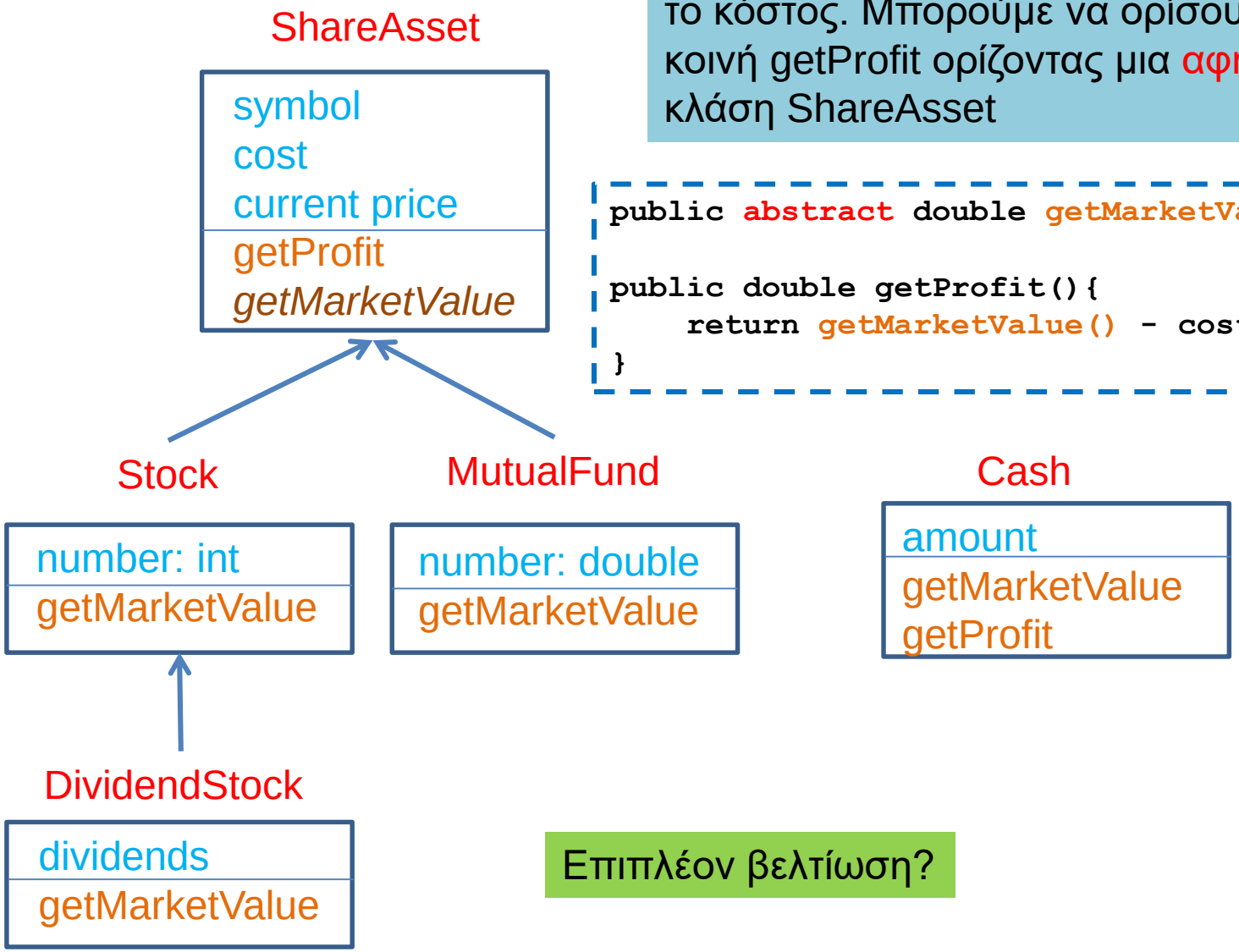
- Βλέπουμε ότι υπάρχουν διάφορα κοινά στοιχεία μεταξύ των διαφόρων οντοτήτων που μας ενδιαφέρουν
  - Χρειαζόμαστε για κάθε asset μια συνάρτηση που να μας δίνει το market value και μία που να υπολογίζει το profit
  - Για τα share assets (stocks, dividend stocks, mutual funds) το κέρδος είναι η διαφορά της τωρινής τιμής με το κόστος
  - Η τιμή των dividend stocks υπολογίζεται όπως αυτή την απλών stocks απλά προσθέτουμε και το μέρισμα

Η DividentStock έχει τα ίδια χαρακτηριστικά με την Stock και απλά αλλάζει ο τρόπος που υπολογίζεται η αποτίμηση ώστε να προσθέτει τα dividents



Πως αλλιώς μπορούμε να βελτιώσουμε το σχεδιασμό?

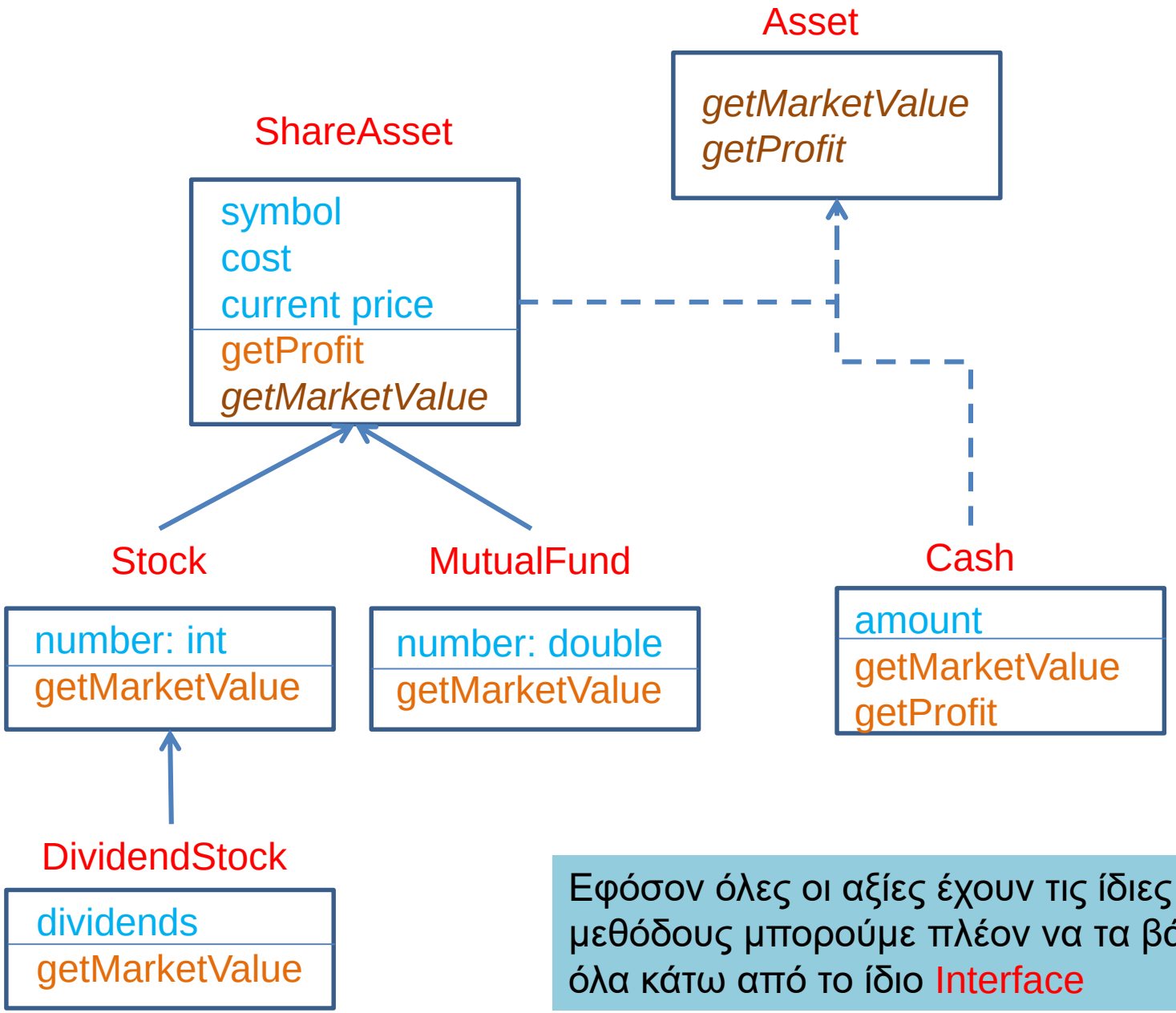
Η getProfit είναι ουσιαστικά η ίδια για όλα τα shares: τωρινή αποτίμηση μείον το κόστος. Μπορούμε να ορίσουμε μια κοινή getProfit ορίζοντας μια αφηρημένη κλάση ShareAsset




```
public abstract double getMarketValue();

public double getProfit(){
     return getMarketValue() - cost;
}
```

Επιπλέον βελτίωση?



Εφόσον όλες οι αξίες έχουν τις ίδιες μεθόδους μπορούμε πλέον να τα βάλουμε όλα κάτω από το ίδιο Interface

```
public interface Asset
{
    public double getMarketValue();

    public double getProfit();
}
```

```
public abstract class ShareAsset implements Asset
{
        private String symbol;
        private double cost = 0;
        private double currentPrice;

        public ShareAsset(String symbol, double price){
                this.symbol = symbol;
                currentPrice = price;
        }

        public abstract double getMarketValue();

        public double getProfit(){
                return getMarketValue() - cost;
        }

        public void setCost(double cost){ this.cost += cost;}

        public void setCurrentPrice(double price){currentPrice = price;}

        public double getCost(){ return cost; }

        public double getCurrentPrice(){return currentPrice;}

        public String getSymbol(){return symbol;}
}
```

```
public class Stock extends ShareAsset
{
        private int number = 0;

        public Stock(String symbol, int number, double costPrice){
                super(symbol,costPrice);
                this.number = number;
                setCost(number*costPrice);
        }

        public Stock(String symbol, double costPrice){
                super(symbol,costPrice);
        }

        public void purchase(int number, double price){
                this.number += number;
                setCost(number*price+getCost());
        }

        public double getMarketValue(){
                return number*getCurrentPrice();
        }

        public int getNumber(){return number;}

        public String toString(){
                return getSymbol() +": " + number + " cost:" + getCost()
                        + "\nCurrent price: " + getCurrentPrice()
                        + "\nMarket Value: " + getMarketValue()
                        + "\nProfit: " + getProfit();
        }
}
```

```
public class MutualFund extends ShareAsset
{
        private double number = 0;

        public MutualFund(String symbol, double number, double costPrice){
                super(symbol,costPrice);
                this.number = number;
                setCost(number*costPrice);
        }

        public MutualFund(String symbol, double costPrice){
                super(symbol,costPrice);
        }

        public void purchase(double number, double price){
                this.number += number;
                 setCost(number*price+getCost());
        }

        public double getMarketValue(){
                return number*getCurrentPrice();
        }

        public double getNumber(){return number;}

        public String toString(){
                return getSymbol() +": " + number + " cost:" + getCost()
                        + "\nCurrent price: " + getCurrentPrice()
                        + "\nMarket Value: " + getMarketValue()
                        + "\nProfit: " + getProfit();
        }
}
```

```
public class DividendStock extends Stock
{
        private double dividends = 0;

        public DividendStock(String symbol, int number, double costPrice){
                super(symbol,number, costPrice);
        }

        public DividendStock(String symbol, double costPrice){
                super(symbol,costPrice);
        }

        public void payDividends(double amountPerShare){
                dividends += amountPerShare*getNumber();
        }

        public double getMarketValue(){
                return super.getMarketValue() + dividends;
        }

        public String toString(){
                return super.toString() +"\nDividends: " + dividends;
        }
}
```

Κλήση των μεθόδων της γονικής κλάσης με χρήση της λέξης super

```
public class Cash implements Asset
{
	private double amount = 0;

	public Cash(double amount)
	{
		this.amount = amount;
	}

	public double getMarketValue(){
		return amount;
	}

	public double getProfit(){
		return 0;
	}

	public String toString(){
		return "Cash: " + amount;
	}
}
```

```
import java.util.*;

public class Portofolio
{
    public static void main(String[] args){
        ArrayList<Asset> myPortofolio = new ArrayList<Asset>();
        myPortofolio.add(new Cash(1000));

        DividendStock msft = new DividendStock("MSFT", 100, 39.5);
        myPortofolio.add(msft);
        msft.setCurrentPrice(40);
        msft.payDividends(0.5);

        MutualFund fund = new MutualFund("FUND", 10.5, 30);
        myPortofolio.add(fund);
        fund.setCurrentPrice(40);
        fund.purchase(3.5, 40);

        Stock appl = new Stock("APPL", 10, 100);
        myPortofolio.add(appl);
        appl.setCurrentPrice(97);

        double totalValue = 0;
        double totalProfit = 0;
        for (Asset a:myPortofolio){
                System.out.println(a+"\n");
                totalValue += a.getMarketValue();
                totalProfit += a.getProfit();
        }
        System.out.println("\nTotal value = "+ totalValue);
        System.out.println("Total profit = "+ totalProfit);
    }
}
```

Χρήση του Interface Asset

Χρήση των μεθόδων του Interface

```
public class Broker
{
    public static void main(String[] args){
        Asset[] portofolio = new Asset[4];
        portofolio[0] = new Cash(20000);
        portofolio[1] = new Stock("GOOG", 100, 800);
        portofolio[2] = new MutualFund("Fund", 10.5, 54.3);
        portofolio[3] = new DividendStock("APPL",200, 900);

        Stock goog = (Stock)portofolio[1];
        goog.setCurrentPrice(900);
        MutualFund fund = (MutualFund)portofolio[2];
        fund.setCurrentPrice(50);
        DividendStock appl = (DividendStock)portofolio[3];
        appl.setCurrentPrice(1000);
        appl.payDividends(0.5);

        double totalValue = 0;
        double totalProfit = 0;
        for (int i = 0; i < 4; i ++){
            System.out.println(portofolio[i]);
            totalValue += portofolio[i].getMarketValue();
            totalProfit += portofolio[i].getProfit();
        }
        System.out.println("Total Value = "+totalValue);
        System.out.println("Total profit = "+totalProfit);
    }
}
```

Χρειαζόμαστε downcasting για να κάνουμε χρήση των μεθόδων