ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Πολυμορφισμός – Αφηρημένες κλάσεις

Interfaces (διεπαφές)

Βρείτε τα λάθη

- Στο πρόγραμμα στην επόμενη διαφάνεια υπάρχουν διάφορα λάθη
  - Ποια είναι?

```java
public abstract class Vehicle
{
  private int position = 0;

  public Vehicle(int pos){
        position = pos;
  }

  public abstract void move();

  public void print()
  {
    System.out.println("position = "
                       + position);
  }
}
```

```java
public class Example
{
  public static void main(String[] args){
    Vehicle[] V = new Vehicle[3];
    V[0] = new Car(0,100);
    V[1] = new Bike();
    V[0].drive(); V[0].print();
    V[1].move(); V[1].print();
    V[2] = new Vehicle(0);
  }
}
```

```java
public class Car extends Vehicle
{
  private int gas;

  public Car(int pos, int gas){
    position = pos;
    this.gas = gas;
  }

  public void drive(){
    position += 10;
    gas -= 10;
  }

  public void print(){
    super.print();
    System.out.println("gas =" + gas);
  }
}
```

```java
public class Bike extends Vehicle
{
  public void move(){
        position ++;
  }
}
```

```
public abstract class Vehicle
{
   private int position = 0;

   public Vehicle(int pos){
         position = pos;
   }

   public abstract void move();

   public void print()
   {
     System.out.println("position = "
                          + position);
   }
}
```

```
public class Example
{
   public static void main(String[] args){
     Vehicle[] V = new Vehicle[3];
     V[0] = new Car(0,100);
     V[1] = new Bike();
     V[0].drive(); V[0].print();
     V[1].move(); V[1].print();
     V[2] = new Vehicle(0);
   }
}
```

```
public class Car extends Vehicle
{
   private int gas;

   public Car(int pos, int gas){
     position = pos;
     this.gas = gas;
   }

   public void drive(){
     position += 10;
     gas -= 10;
   }

   public void print(){
     super.print();
     System.out.println("gas =" + gas);
   }
}
```

```
public class Bike extends Vehicle
{
   public void move(){
         position ++;
   }
}
```

```
public abstract class Vehicle
{
   protected int position = 0;

   public Vehicle(){
   }

   public Vehicle(int pos){
       position = pos;
   }

   public abstract void move();

   public void print()
   {
     System.out.println("position = "
                        + position);
   }
}
```

Το πεδίο position πρέπει να είναι protected εφόσον το χρησιμοποιούν και οι παράγωγες κλάσεις

Πρέπει να ορίσουμε και ένα κενό constructor, ή να καλούμε την super μέσα στις παράγωγες κλάσεις.

```
public class Car extends Vehicle
{
  private int gas;

  public Car(int pos, int gas){
    position = pos;
    this.gas = gas;
  }

  public void move(){
    position += 10;
    gas -= 10;
  }

  public void print(){
    super.print();
    System.out.println("gas =" + gas);
  }
}
```

Ο constructor δουλεύει μόνο αν έχουμε constructor χωρίς ορίσματα στην Vehicle. Αλλιώς χρειαζόμαστε αυτό τον constructor:

```
public Car(int pos, int gas){
   super(pos);
   this.gas = gas;
 }
```

Η Car πρέπει να υλοποίει την μέθοδο move

```
public class Bike extends Vehicle
{
  public void move(){
     position ++;
  }
}
```

Ο constructor (ή μάλλον η έλλειψη του) δουλεύει μόνο αν έχουμε constructor χωρίς ορίσματα στην Vehicle. Αλλιώς χρειαζόμαστε αυτό τον constructor:

```
 public Bike(){
    super(0);
}
```

```
public class Example
{
  public static void main(String[] args){
    Vehicle[] V = new Vehicle[2];
    V[0] = new Car(0,100);
    V[1] = new Bike();
    V[0].move(); V[0].print();
    V[1].move(); V[1].print();
    // V[2] = new Vehicle(0);
  }
}
```

Δεν μπορούμε να ορίσουμε αντικείμενο τύπου Vehicle γιατί είναι αφηρημένη κλάση.

Ερωτήσεις:

- Υπάρχει πρόβλημα με την εντολή `Vehicle[] V = new Vehicle[2];` ?
- Ποια print καλείται για το αντικείμενο V[0]? Ποια για το V[1]? Γιατί?
- Τι θα τυπώσει το πρόγραμμα?

Υπάρχει κάποιο λάθος σε αυτό τον ορισμό?

```
public abstract class EngineVehicle extends Vehicle
{
      protected int gas;

      public EngineVehicle(int pos, int gas){
         super(pos);
         this.gas = gas;
      }
}
```

Όχι. Εφόσον η EngineVehicle είναι αφηρημένη δεν χρειάζεται να ορίσουμε την αφηρημένη μέθοδο move

INTERFACES (ΔΙΕΠΑΦΕΣ)

Αφηρημένες κλάσεις

- Αφηρημένες κλάσεις είναι οι κλάσεις που περιέχουν αφηρημένες μεθόδους
  - Η υλοποίηση των αφηρημένων μεθόδων μετατίθεται στις μη αφηρημένες (ενυπόστατες – concrete) κλάσεις που είναι απόγονοι μιας αφηρημένης κλάσης.
  - Η υλοποίηση είναι υποχρεωτική. Άρα έτσι εξασφαλίζουμε ότι μια concrete κλάση θα έχει την μέθοδο που θέλουμε.
- Οι αφηρημένες κλάσεις εκτός από αφηρημένες μεθόδους έχουν και πεδία και ενυπόστατες μεθόδους.
  - Κληρονομούν επιπλέον χαρακτηριστικά στους απογόνους τους, όχι μόνο τις αφηρημένες μεθόδους.

Interfaces

- Ένα interface είναι μια ακραία μορφή αφηρημένης κλάσης
  - Ένα interface έχει μόνο δηλώσεις μεθόδων.
  - Το interface ορίζει μια απαραίτητη λειτουργικότητα που θέλουμε.

Παραδείγματα

```
public interface MovingObject
{
      public void move();
}
```

```
public interface ElectricObject
{
      public boolean powerOn();

      public boolean powerOff();
}
```

Interfaces

- Μία κλάση υλοποιεί το interface.
  - Η κλάση μπορεί να είναι και αφηρημένη κλάση
- Μια κλάση μπορεί να υλοποιεί πολλαπλά interfaces
  - Αλλά δεν μπορεί να κληρονομεί από πολλαπλές κλάσεις

Παραδείγματα

```
public class Car implements MovingObject
{
        …
}
```

```
public class ElectricCar
        implements MovingObject, ElectricObject
{       …       }
```

```
public abstract class Vehicle implements MovingObject
{
        public abstract void move();
}
```

```
public class ElectricCar
        extends Vehicle, implements ElectricObject
{       …       }
```

Interfaces

- Ένα Interface μπορεί να κληρονομεί από ένα άλλο interface

```
public interface ElectricMovingObject
      extends MovingObject
{
      public boolean powerOn();

      public boolean powerOff();
}
```

Interfaces vs αφηρημένες κλάσεις

- Τα interfaces είναι χρήσιμα όταν θέλουμε να ορίσουμε αντικείμενα που ορίζονται μόνο από κάποια υψηλού επιπέδου λειτουργικότητα ενώ κατά τα άλλα μπορεί να είναι πολύ διαφορετικά μεταξύ τους
  - Έχουν το ίδιο interface – ένα κινούμενο αντικείμενο μπορεί να κινείται
    - Δεν ξέρουμε πως, σε πόσες διαστάσεις, με τι ταχύτητα κλπ.
- Μια αφηρημένη κλάση υποθέτει ότι τα αντικείμενα που θα ορίσουμε έχουν πολλά περισσότερα κοινά χαρακτηριστικά
  - Κοινά πεδία πάνω στα οποία μπορούμε να υλοποιήσουμε και κοινές μεθόδους.

Αφηρημένοι Τύποι Δεδομένων

- Τα interfaces μπορούμε να τα δούμε και σαν Αφηρημένους Τύπους Δεδομένων
- Π.χ., μία στοίβα απαιτεί συγκεκριμένες λειτουργίες από τις κλάσεις που την υλοποιούν
  - Push
  - Pop
  - IsEmpty
  - Top
- Ανάλογα με τον τύπο των δεδομένων που θα κρατάει η στοίβα μπορούμε να ορίσουμε διαφορετικές υλοποιήσεις
  - Υπάρχει και άλλος τρόπος να το κάνουμε αυτό όμως όπως θα δούμε παρακάτω

Παράδειγμα: Το interface Comparable

- Το interface Comparable είναι ένα υπάρχον interface το οποίο ορίζει διεπαφή για αντικείμενα τα οποία μπορούν να συγκριθούν μεταξύ τους
- Ορίζει την μέθοδο
  - `public int compareTo(Object other);`
- Σημασιολογία:
  - Αν η μέθοδος επιστρέψει αρνητικό αριθμό τότε το αντικείμενο this είναι μικρότερο από το αντικείμενο other
  - Αν η μέθοδος επιστρέψει μηδέν τότε το αντικείμενο this είναι ίσο με το αντικείμενο other
  - Αν η μέθοδος επιστρέψει θετικό αριθμό τότε το αντικείμενο this είναι μεγαλύτερο από το αντικείμενο other

Εφαρμογή

- Μπορούμε να ορίσουμε μια μέθοδο sort η οποία να μπορεί να εφαρμοστεί σε πίνακες με οποιαδήποτε μορφής αντικείμενα

```
public static void sort(Comparable[] array){
  for (int i = 0; i < array.length; i ++){
    Comparable minElement = array[i];
    for (int j = i+1; j < array.length; j ++){
      if (minElement.compareTo(array[j]) > 0){
       minElement = array[j];
       array[j] = array[i];
       array[i] = minElement;
      }
    }
  }
}
```

```
import java.util.Scanner;

class Person implements Comparable
{
  private String name;
  private int number;

  public Person(){
        System.out.println("enter name and number:");
        Scanner input = new Scanner(System.in);
        name = input.next(); number = input.nextInt();
  }

  public String toString(){
        return name + " " + number;
  }

  public int compareTo(Object other){
        Person otherPerson = (Person) other;
        if (number < otherPerson.number){
                return -1;
        }else if (number == otherPerson.number){
                return 0;
        } else { return 1;}
  }
}
```

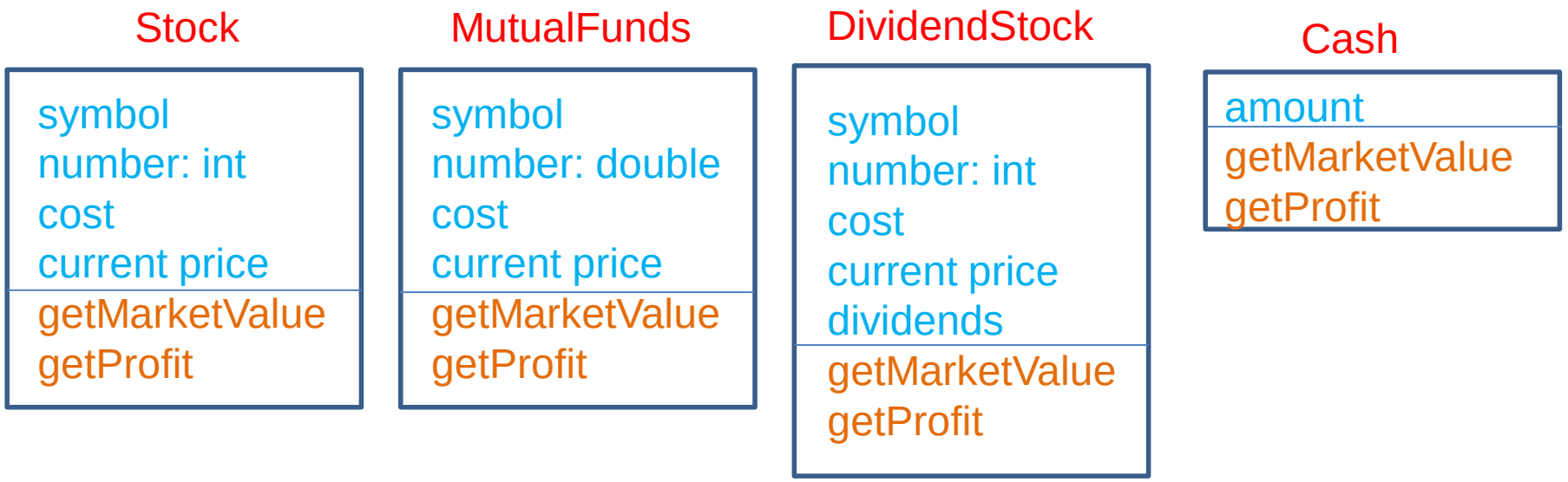
```
public class ComparableExample
{
  public static void main(String[] args){
    Person[] array = new Person[5];
    for (int i = 0; i < array.length; i ++){
        array[i] = new Person();
    }
    sort(array);
    System.out.println();
    for (int i = 0; i < array.length; i ++){
        System.out.println(array[i]);
    }
  }

  public static void sort(Comparable[] array){
    for (int i = 0; i < array.length; i ++){
        Comparable minElement = array[i];
        for (int j = i+1; j < array.length; j ++){
          if (minElement.compareTo(array[j]) > 0){
                minElement = array[j];
                array[j] = array[i];
                array[i] = minElement;
          }
        }
    }
  }
}
```

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

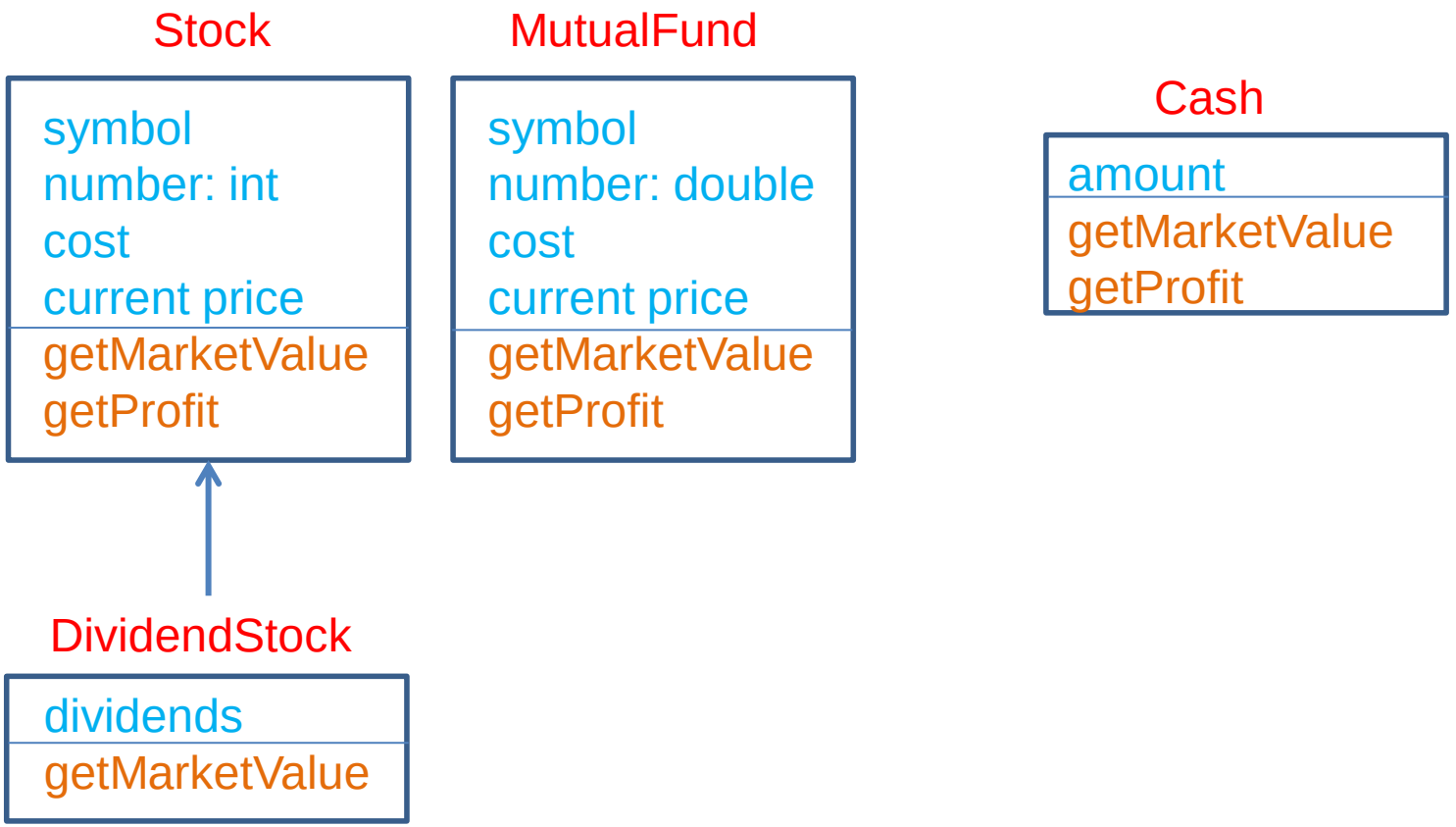
Λεπτομέρειες

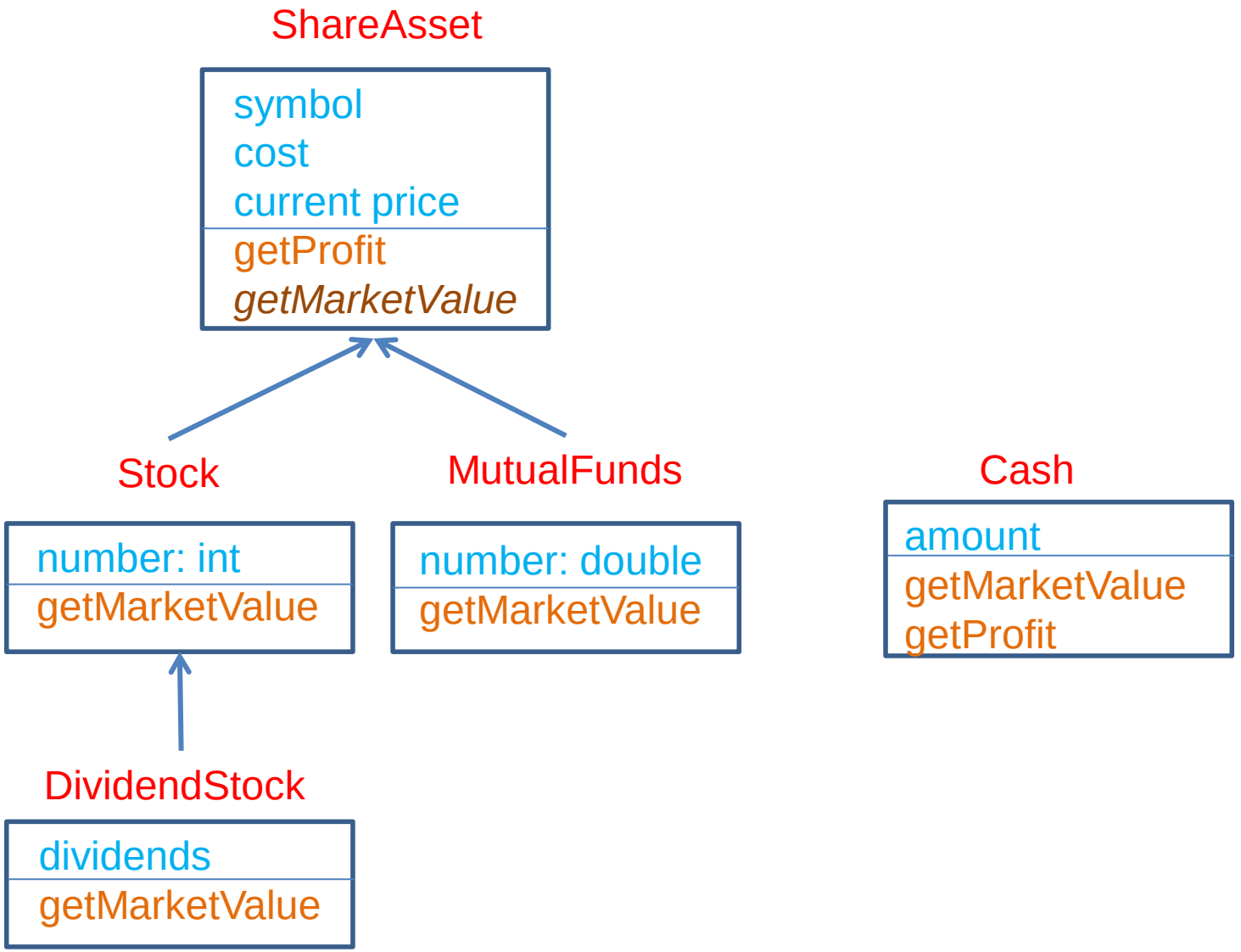
- Cash: Δεν μεταβάλλεται η αξία του, δεν έχει κέρδος
- Stocks: Η αξία του είναι ίση με τον αριθμό των μετοχών επί την αξία της μετοχής. Το κέρδος είναι η διαφορά της τωρινής αποτίμησης με το κόστος αγοράς
- Mutual Funds: Παρόμοια με τα Stocks αλλά ο αριθμός των μετοχών που μπορούμε να έχουμε είναι πραγματικός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

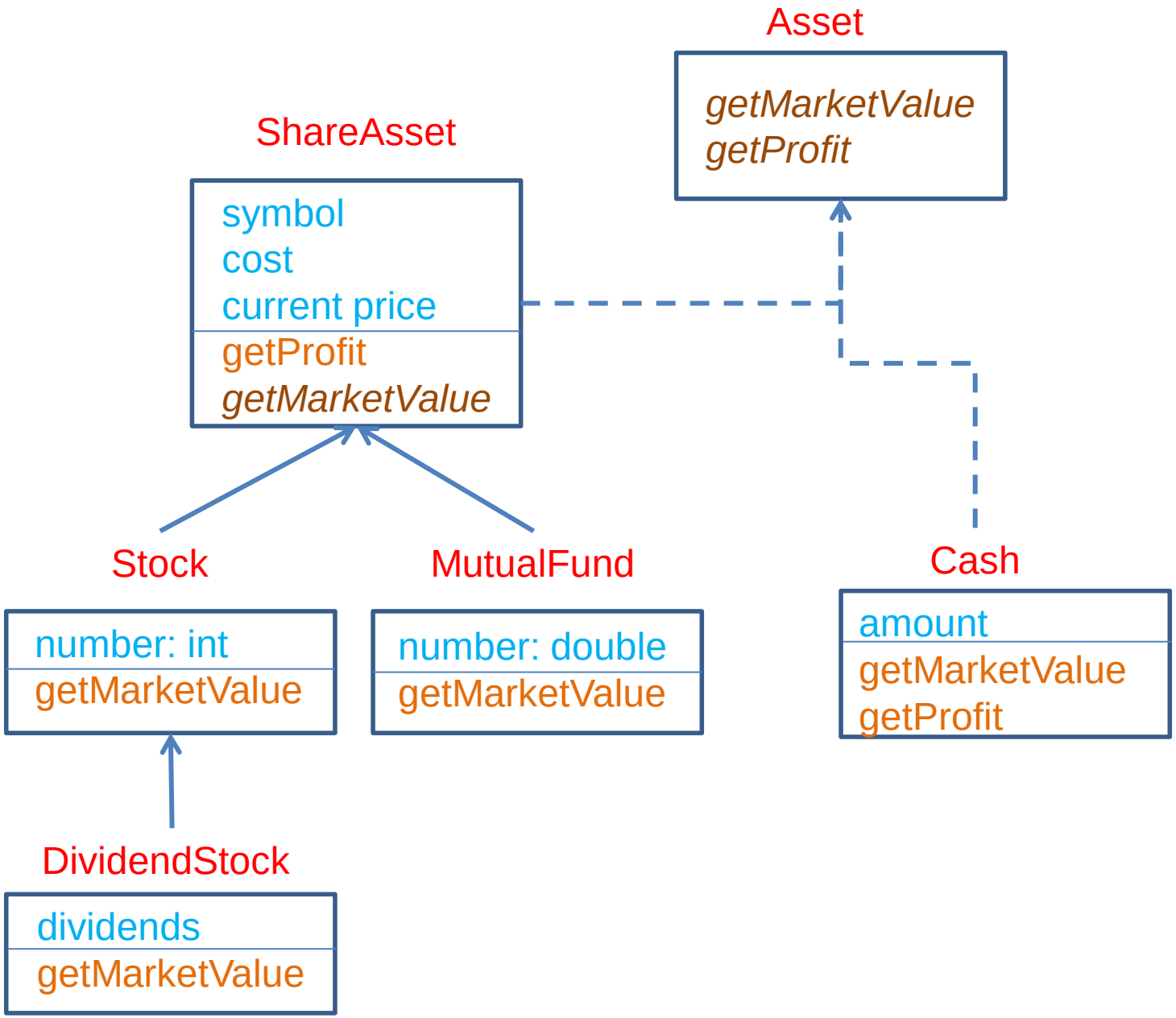

Stock
symbol
number: int
cost
current price
getMarketValue
getProfit
MutualFunds
symbol
number: double
cost
current price
getMarketValue
getProfit
DividendStock
symbol
number: int
cost
current price
dividends
getMarketValue
getProfit
Cash
amount
getMarketValue
getProfit

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


Stock
symbol
number: int
cost
current price
getMarketValue
getProfit
MutualFund
symbol
number: double
cost
current price
getMarketValue
getProfit
Cash
amount
getMarketValue
getProfit
DividendStock
dividends
getMarketValue


ShareAsset
symbol
cost
current price
getProfit
getMarketValue
Stock
MutualFunds
Cash
number: int
getMarketValue
number: double
getMarketValue
amount
getMarketValue
getProfit
DividendStock
dividends
getMarketValue


Asset
getMarketValue
getProfit
ShareAsset
symbol
cost
current price
getProfit
getMarketValue
Stock
number: int
getMarketValue
MutualFund
number: double
getMarketValue
Cash
amount
getMarketValue
getProfit
DividendStock
dividends
getMarketValue