COSC 1020

Yves Lespérance

Lecture Notes

Week 8 — Composition & Collections

Recommended Readings:
Horstmann: Ch. 16 Sec. 3
Lewis & Loftus: Ch. 4 Sec. 6 and Ch. 10 Sec. 3

Composition

Computer applications often involve complex structured data.

One very important data structuring mechanism is **composition** — one piece of data is composed of subparts, it **has a** second piece of data as a component. E.g. a `Stock` has a symbol, an `Investment` has a `Stock`.

The more interesting cases are when an object is composed of other objects.

In math, corresponding structuring mechanism is Cartesian product.

In Java, objects are the main composition mechanism.

Can use UML class diagrams to represent relationships between classes including composition. E.g. `Investment`, `VehicleTransfer`.

Other terms used for the composition/has-a relationship include aggregation, part-of, association. Can make a finer analysis.

E.g. Using `Investment`

```
import type.lang.*;
import type.lib.*;
public class InvestmentTest
{  public static void main(String[] args)
   {  Stock stk1 = new Stock("NT");
      IO.println("stk1= " + stk1.toString());
      Investment inv1 = new Investment(stk1,100,
                                  stk1.getPrice());
      IO.println("inv1= " + inv1.toString());
      IO.println("inv1.getQty()= " + inv1.getQty());
      IO.println("inv1.getBookValue()= " +
                 inv1.getBookValue());
      IO.println("inv1.getStock().toString()= " +
                 inv1.getStock().toString());
      IO.println("inv1.getStock().getSymbol()= " +
                 inv1.getStock().getSymbol());
      Stock stk2 = new Stock("BMO");
      IO.println("stk2= " + stk2.toString());
      Investment inv2 = new Investment(stk2,100,
                                   stk2.getPrice());
      IO.println("inv2= " + inv2.toString());
      IO.println("inv1.equals(inv2)= " +
                 inv1.equals(inv2));
      Stock stk3 = inv1.getStock();
      IO.println("stk3= " + stk3.toString());
      stk3.setSymbol("RY");
      IO.println("reset stk3’s symbol to RY");
      IO.println("stk3= " + stk3.toString());
      IO.println("inv1.getStock()= " +
                 inv1.getStock().toString());
      stk2.setSymbol("RY");
      IO.println("reset stk2’s symbol to RY");
      IO.println("stk2= " + stk2.toString());
      IO.println("inv2.getStock()= " +
                 inv2.getStock().toString());
      IO.println("inv1.equals(inv2)= " +
                 inv1.equals(inv2));
   }
}
```

```
zebra 346 % java InvestmentTest
stk1= NT Nortel Networks Corp
inv1= NT Nortel Networks Corp QTY=100 BV=1.98
inv1.getQty()= 100
inv1.getBookValue()= 1.98
inv1.getStock().toString()= NT Nortel Networks Corp
inv1.getStock().getSymbol()= NT
stk2= BMO Bank of Montreal
inv2= BMO Bank of Montreal QTY=100 BV=40.45
inv1.equals(inv2)= false
stk3= NT Nortel Networks Corp
reset stk3’s symbol to RY
stk3= RY Royal Bank of Canada
inv1.getStock()= RY Royal Bank of Canada
reset stk2’s symbol to RY
stk2= RY Royal Bank of Canada
inv2.getStock()= RY Royal Bank of Canada
inv1.equals(inv2)= false
zebra 347 %
```

As we can see, `Investment` constructor uses `Stock` object passed as component of the `Investment` and `getStock` accessor method returns a reference to this component. This allows a user to change the state of the `Investment` object with strange results.

`Investment` could protect against this by setting its component to a copy of the passed `Stock` in the constructor and returning a reference to a copy of its component in `getStock`. Then, only methods belonging to `Investment` could change the `Stock` component.

In general, to ensure that users cannot change components of an object without using the object’s methods, one must make a *deep copy* of components passed to/from the object, i.e. a copy where subcomponents and subsubcomponents are also copied.

Collections

In many case, an object has a whole *collection* of components, e.g. a `Portfolio` has a collection of `Investments`, a `Course` has a collection of `Students`.

In math, the corresponding structuring mechanism is sets.

In Java, there are several mechanisms to deal with collections, in particular, arrays and the `Vector` class. More about these later. Some classes, e.g. `Portfolio`, hide/encapsulate the mechanism used.

Can use UML class diagrams to represent *having a collection of components*.

Iteration over Collections

Often you need to do some operations on each element of a collection. This is called *iterating* over the collection.

Objects that have collections of components provide mechanisms for this.

E.g. the `Portfolio` class provides two groups of methods for iterating over the `Investments` in the portfolio:

`getInvestment(`*i*`)` together with `getCount()`and

`getFirst()` together with `getNext()`.

`getNext()` can be viewed as an iterator that is initialized by `getFirst()`.

```
import type.lang.*;
import type.lib.*;
public class PortfolioTest
{  public static void main(String[] args)
   {  Portfolio ptf1 = new Portfolio("My e.g. Portfolio",10);
      IO.println(ptf1.toString());
      Stock stk1 = new Stock("NT");
      ptf1.add(new Investment(stk1,100,stk1.getPrice()));
      stk1 = new Stock("BMO");
      ptf1.add(new Investment(stk1,50,stk1.getPrice()));
      stk1 = new Stock("RY");
      ptf1.add(new Investment(stk1,100,stk1.getPrice()));
      IO.println(ptf1.toString());
      IO.println("Iterate over investments using getInvestment");
      int ptf1count = ptf1.getCount();
      for(int i = 0; i < ptf1count; i++)
      {  Investment inv = ptf1.getInvestment(i);
         IO.println(inv);
      }
      IO.println("Iterate over investments using getFirst/Next");
      for(Investment inv = ptf1.getFirst();
          inv != null; inv = ptf1.getNext())
      {  IO.println(inv);
      }
```

```
      stk1 = new Stock("NT");
      ptf1.add(new Investment(stk1,150,stk1.getPrice()));
      IO.println("Iterate again over investments using getFirst/Ne
      for(Investment inv = ptf1.getFirst();
          inv != null; inv = ptf1.getNext())
      {  IO.println(inv);
      }
      IO.println(ptf1.toString());
   }
}
```

```
blue 318 % java PortfolioTest
My e.g. Portfolio: 0
My e.g. Portfolio: 3
Iterate over investments using getInvestment
NT Nortel Networks Corp QTY=100 BV=1.71
BMO Bank of Montreal QTY=50 BV=39.03
RY Royal Bank of Canada QTY=100 BV=55.1
Iterate over investments using getFirst/Next
NT Nortel Networks Corp QTY=100 BV=1.71
BMO Bank of Montreal QTY=50 BV=39.03
RY Royal Bank of Canada QTY=100 BV=55.1
Iterate again over investments using getFirst/Next
NT Nortel Networks Corp QTY=100 BV=1.71
BMO Bank of Montreal QTY=50 BV=39.03
RY Royal Bank of Canada QTY=100 BV=55.1
NT Nortel Networks Corp QTY=150 BV=1.71
My e.g. Portfolio: 4
blue 319 %
```

A `Portfolio` Can be Modified in Strange Ways

```
import type.lang.*;
import type.lib.*;
public class PortfolioTest2
{  public static void main(String[] args)
   {  Portfolio ptf1 = new Portfolio("My e.g. Portfolio",10);
      Stock stk1 = new Stock("NT");
      ptf1.add(new Investment(stk1,100,stk1.getPrice()));
      stk1 = new Stock("BMO");
      ptf1.add(new Investment(stk1,50,stk1.getPrice()));
      stk1 = new Stock("RY");
      ptf1.add(new Investment(stk1,100,stk1.getPrice()));
      IO.println(ptf1.toString());
      for(Investment inv = ptf1.getFirst(); inv != null;
          inv = ptf1.getNext())
      {  IO.println(inv);
      }
      IO.println();
      Investment inv1 = ptf1.getFirst();
      stk1 = inv1.getStock();
      stk1.setSymbol("RY");
      IO.println("Changed 1st investment to RY");
      IO.println(ptf1.toString());
      for(Investment inv = ptf1.getFirst(); inv != null;
          inv = ptf1.getNext())
      {  IO.println(inv);
      }
   }
}
```

```
blue 310 % java PortfolioTest2
My e.g. Portfolio: 3
NT Nortel Networks Corp QTY=100 BV=1.9
BMO Bank of Montreal QTY=50 BV=38.1
RY Royal Bank of Canada QTY=100 BV=54.41

Changed 1st investment to RY
My e.g. Portfolio: 3
RY Royal Bank of Canada QTY=100 BV=1.9
BMO Bank of Montreal QTY=50 BV=38.1
RY Royal Bank of Canada QTY=100 BV=54.41
blue 311 %
```

Making a Deep Copy of a Portfolio

```
import type.lang.*;
import type.lib.*;
public class PortfolioDeepCopy
{  public static void main(String[] args)
   {  Portfolio ptf1 = new Portfolio("My e.g. Portfolio",10);
      Stock stk1 = new Stock("NT");
      ptf1.add(new Investment(stk1,100,stk1.getPrice()));
      stk1 = new Stock("BMO");
      ptf1.add(new Investment(stk1,50,stk1.getPrice()));
      stk1 = new Stock("RY");
      ptf1.add(new Investment(stk1,100,stk1.getPrice()));
      IO.println("\nPortfolio ptf1 is:");
      IO.println(ptf1.toString());
      for(Investment inv = ptf1.getFirst(); inv != null;
          inv = ptf1.getNext())
      {  IO.println(inv);
      }
      // Make deep copy
      Portfolio ptf1c = new Portfolio(ptf1.getName(),10);
      for(Investment inv = ptf1.getFirst(); inv != null;
          inv = ptf1.getNext())
      {  Stock stkc = new Stock(inv.getStock().getSymbol());
         Investment invc = new Investment(stkc,
            inv.getQty(),inv.getBookValue());
         ptf1c.add(invc);
      }
      IO.println("\nMade deep copy ptf1c of ptf1:");
      IO.println(ptf1c.toString());
      for(Investment inv = ptf1c.getFirst(); inv != null;
          inv = ptf1c.getNext())
      {  IO.println(inv);
      }
```

```
      // Change 1st investment in ptf1 to RY
      Investment inv1 = ptf1.getFirst();
      stk1 = inv1.getStock();
      stk1.setSymbol("RY");
      IO.println("\nChanged 1st investment in ptf1 to RY");
      IO.println("\nptf1 is now:");
      IO.println(ptf1.toString());
      for(Investment inv = ptf1.getFirst(); inv != null;
          inv = ptf1.getNext())
      {  IO.println(inv);
      }
      IO.println("\nptf1c is now:");
      IO.println(ptf1c.toString());
      for(Investment inv = ptf1c.getFirst(); inv != null;
          inv = ptf1c.getNext())
      {  IO.println(inv);
      }
   }
}
```

```
blue 316 % java PortfolioDeepCopy

Portfolio ptf1 is:
My e.g. Portfolio: 3
NT Nortel Networks Corp QTY=100 BV=1.9
BMO Bank of Montreal QTY=50 BV=38.1
RY Royal Bank of Canada QTY=100 BV=54.41

Made deep copy ptf1c of ptf1:
My e.g. Portfolio: 3
NT Nortel Networks Corp QTY=100 BV=1.9
BMO Bank of Montreal QTY=50 BV=38.1
RY Royal Bank of Canada QTY=100 BV=54.41

Changed 1st investment in ptf1 to RY

ptf1 is now:
My e.g. Portfolio: 3
RY Royal Bank of Canada QTY=100 BV=1.9
BMO Bank of Montreal QTY=50 BV=38.1
RY Royal Bank of Canada QTY=100 BV=54.41

ptf1c is now:
My e.g. Portfolio: 3
NT Nortel Networks Corp QTY=100 BV=1.9
BMO Bank of Montreal QTY=50 BV=38.1
RY Royal Bank of Canada QTY=100 BV=54.41
blue 317 %
```

The Vector Class

The Vector class provides a generic mechanism for maintaining a collection of objects, e.g. a set of favorite Stocks.

Elements are ordered starting from index 0. A Vector can grow and shrink as needed. New elements can be inserted in arbitrary positions and the remaining elements will shift to higher indices.

See the Vector class's API for available methods.

As we can see in the following e.g., a variety of mechanisms are available for iterating over the elements of a Vector.

Note also that the elementAt(i) method's return type is Object. If you want to assign the returned element to a variable of a more specific type, e.g. Stock, you must cast it. Only then can you use methods belonging to the more specific class on the object.

```
import type.lang.*;
import type.lib.*;
import java.util.*;
public class VectorTest
{  public static void main(String[] args)
   {  Vector hotPicks = new Vector();
      IO.println(hotPicks.toString());
      Stock stk1 = new Stock("NT");
      hotPicks.add(stk1);
      stk1 = new Stock("BMO");
      hotPicks.add(stk1);
      stk1 = new Stock("RY");
      hotPicks.add(stk1);
      IO.println(hotPicks.toString());
      int hotPicksSize = hotPicks.size();
      for(int i = 0; i < hotPicksSize; i++)
      {  Stock stk2 = (Stock) hotPicks.elementAt(i);
         stk2.refresh();
         IO.println(stk2.toString() + " " + stk2.getPrice());
      }
      IO.println();
      hotPicks.insertElementAt(new Stock("ALI"),1);
      Iterator hpIter = hotPicks.iterator();
      while(hpIter.hasNext())
      {  Stock stk3 = (Stock) hpIter.next();
         stk3.refresh();
         IO.println(stk3.toString() + " " + stk3.getPrice());
      }
```

```
      IO.println();
      hotPicks.remove(0);
      hotPicks.remove(new Stock("BMO"));
      Enumeration hpEnum = hotPicks.elements();
      while(hpEnum.hasMoreElements())
      {  Stock stk4 = (Stock) hpEnum.nextElement();
         stk4.refresh();
         IO.println(stk4.toString() + " " + stk4.getPrice());
      }

   }
}
```

```
zebra 328 % java VectorTest
[]
[NT Nortel Networks Corp, BMO Bank of Montreal, RY Royal Bank of
NT Nortel Networks Corp 2.11
BMO Bank of Montreal 39.47
RY Royal Bank of Canada 56.7

NT Nortel Networks Corp 2.11
ALI Allican Resources Inc. 0.29
BMO Bank of Montreal 39.47
RY Royal Bank of Canada 56.7

ALI Allican Resources Inc. 0.29
RY Royal Bank of Canada 56.7
zebra 329 %
```