Supervision work

Tudor Avram, Homerton College, tma33@cam.ac.uk

17.11.2015

1 Creating Classes

1.1 Exercise 11

(a) Code for the mutable class :

```
package uk.ac.cam.tma33.s6;

public class Vector2D {
public float x;
public float y;

public Vector2D addVectors(Vector2D a, Vector2D b) {
        Vector2D N = new Vector2D();
        N.x = a.x + b.x;
        N.y = a.y + b.y;
        return N;
}

public float scalarProd(Vector2D a, Vector2D b) {
        return a.x*b.x + a.y*b.y;
}

public float vectorMagnitude(Vector2D a) {
        return (float)Math.sqrt((double)scalarProd(a,a));
}

public Vector2D vectorNorm(Vector2D a){
        Vector2D N = new Vector2D();
        float m = vectorMagnitude(a);
        N.x = a.x/m;
        N.y = a.y/m;
        return N;
}
}
```

(b) In order to make it immutable, I would have made the 2 variables x and y private and I would have added three more methods : *setVector, getX* and *getY* :

```
private float x;
private float y;

public Vector2D setVector(float a,float b) {
          Vector2D N = new Vector2D();
          N.x = a; N.y = b;
          return N;
}

public float getX(Vector2D V) {
         return V.x;
}

public float getY(Vector2D V) {
         return V.y;
}
```

(c)

• *public void add(Vector2D v)* ——— >This approach is cannot add two vectors for 2 reasons : (i) it only takes one argument; (ii) it doesn't return anything.

• *public Vector2D add(Vector2D v)* ——— >This approach is not suitable for adding 2 vectors because, even if it returns a vector, it only takes 1 argument. Therefore, it could at most add a vector to itself.

• *public Vector2D add(Vector2D v1, Vector2D v2)* ——— >This function can return the sum of 2 vectors in both the mutable and immutable versions of Vector2D class, because it has enough arguments and also returns a Vector2D type element.

• *public static Vector2D add(Vector2D v1, Vector2D v2)* ——— >This method will also work just fine.

(d) I could use a special name of the class, for example writing "Immutable" in front of the name of the actual class. (ex. : *"ImmutableVector2D"*)

1.2 Exercise 12

=== OOPLinkedListElement class ===

```
public class OOPLinkedListElement {

public OOPLinkedListElement next;
public int value;
public OOPLinkedListElement() {
         value = 0;
         next = null;
         }
}
```

=== OOPLinkedList class ===

```
package uk.ac.cam.tma33.s6;

public class OOPLinkedList {

protected OOPLinkedListElement head = null;

public int getHead() throws NullPointerException {
         if (head == null) throw new
          NullPointerException("Empty␣linked␣list!");
         return head.value;
}

public void deleteHead() throws NullPointerException {
         if (head == null) throw new
          NullPointerException("Empty␣linked␣list!");
         else head = head.next;
}

public void Insert(int x)  {
         OOPLinkedListElement a = new
                  OOPLinkedListElement();
         if (head!=null)
                  { a.value = x; a.next = head; }
         else
                  { a.value = x; a.next = null;}
         head = a;
}

public int length() {
         int Sol = 0;
         while (head != null) {
                  Sol++;
                  head = head.next;
         }
         return Sol;
}
public int getNth(int n) throws Exception {
        int p = 1;
        while (p < n && head!=null){
                p++;
                head   = head.next;
        }
        if (head == null) throw new
                 Exception("There is no nth element!");
        return head.value;
}

}
```

1.3 Exercise 13

=== The binary tree class ===

```
public class BinaryTree {

public BTNode root;

public BinaryTree(){
        root = null;
}


}
```

It uses the auxiliary class "BTNode" :

```
public class BTNode {

private int value;
private BTNode left, right;

public BTNode() {
        value = 0;
        left = null; right = null;
}

public BTNode(int x) {
        value = x;
        left = null; right = null;
}

public void setLeft(BTNode N){
        left = N;
}

public void setRight(BTNode N){
        right = N;
}

public void setValue(int d) {
        value = d;
}

public BTNode getRight() {
        return right;
}

public BTNode getLeft() {
        return left;
}

public int getValue() {
        return value;
}


}
```

=== The functional array class ===

```
public class FunctionalArray {

private BTNode root;

public FunctionalArray() {
        root = null;
}

public int length(){
        return length(root);
}

private int length(BTNode N){
        if (N != null) return 1 + length(N.getLeft()) +
                 length(N.getRight());
        else return 0;
```

```
}

public void updatePosition(int p, int x) throws Exception{
        updatePosition(p,x,root);
}

private void updatePosition(int p, int x, BTNode N)
                        throws Exception{
        if (p==1) N.setValue(x);
        else if (N == null)
        throw new Exception("No such position!");
        else if(p%2==1)updatePosition(p/2,x,N.getRight());
        else updatePosition(p/2,x,N.getLeft());
}

public void insertValue(int x){
        root = insertValue(x,length(root)+1,root);
}

private BTNode insertValue(int x,int p, BTNode N){
        if (p == 1) N = new BTNode(x);
        else if (p%2==1)
          N.setRight(insertValue(x,p/2,N.getRight()));
        else if (p%2==0)
          N.setLeft(insertValue(x,p/2,N.getLeft()));
        return N;
}

public int getNth(int p) throws Exception{
        return getNth(p,root);
}

private int getNth(int p, BTNode N)throws Exception{
        if (p==1) return N.getValue();
        else if (N == null) throw new
                Exception("There is no such element!");
        else if (p%2 == 0) return getNth(p/2,N.getLeft());
        else return getNth(p/2,N.getRight());
}

public int[] getArray() throws Exception{
        return getArray(root);
}

private int[] getArray(BTNode N) throws Exception{
        if (N == null) throw new
```

```
                Exception("The tree is empty!");
        else {
          int l = length(N);
          int[] ans = new int[l];
          for (int i=0;i<l;i++)
                ans[i] = getNth(i+1,N);
          return ans;
          }
}

public void deleteFirst() throws Exception{
        root = deleteFirst(root);
}

private BTNode deleteFirst(BTNode N) throws Exception{
        if (N == null) throw new
                Exception("The tree is empty!");
        else {
          BTNode Lt = N.getLeft();
          BTNode Rt = N.getRight();
          if (Lt == null && Rt == null) N = null;
          else {
            N.setLeft(Rt);
            int aux = N.getValue();
            N.setValue(Lt.getValue());
            Lt.setValue(aux);
            N.setRight(deleteFirst(Lt));
            }
          return N;
         }
}

}
```

2 Inheritance

2.1 Exercise 14

3D vectors do not act like 2D ones when there are acted methods upon it. For example, the sum of 2 vectors requires 2 coordonates for 2D vectors and 3 for 3D vectors. Therefore, a 3D vector may be a ”2D vector with some added” in a pure mathematical view, but from a programmer’s point of view this argument is wrong. That is because all the methods of Vector2D class have to be rewritten in Vector3D, because they need to take different arguments. So we can deduce that Vector3D deriving from Vector2D is not a mistake, but it is just a useless action to make.

2.2 Exercise 15

Let's consider the following class :

```
package uk.ac.cam.tma33.s6;

public class Access {
        int x = 10;
}
```

When we don't specify the access modifier when we declare a member field of a class, it is automatically defined as being visible only to the package it belongs to. This fact can be verified through the following code :

```
package uk.ac.cam.tma33.s6

public class Test {
        static public void main(String[] args) {
                Access a = new Access();
                System.out.println(a.x);
        }
}
```

which works just fine, printing "10". =¿ It gives no error, as the following code would :

```
package uk.ac.cam.tma33.test;
import uk.ac.cam.tma33.s6.Access;

public class Test {
        static public void main(String[] args) {
                Access a = new Access();
                System.out.println(a.x);
        }
}
```

Therefore, we can deduce that, if we don't specify the access modifier for a class' member, it can only be accessed in the same package as the function.

2.3 Exercise 16

WILL SUBMIT LATER TODAY, WHEN I GET TO THE COLLEGE AND CAN SCAN IT

2.4 Exercise 17

(i) When *MODIFIER = public* :
  (a), (b), (c), (d) cases WORK(compile + no errors);
(ii) When *MODIFIER = private* none of the cases compile.
(iii) When *MODIFIER is unspecified* :
  (a) and (c) WORK (compile and don't give errors at runtime)
  (b) and (d) give compiler error.
(iv) When *MODIFIER = protected* :
  (a), (b) and (c) WORK (compile and don't give errors at runtime)
  (d) gives compiler error.

2.5 Exercise 18

```java
package uk.ac.cam.tma33.s6;

public class OOPSortedLinkedList extends OOPLinkedList{

@Override
public void Insert(int x) {
   OOPLinkedListElement a = new OOPLinkedListElement();
   OOPLinkedListElement current = new
            OOPLinkedListElement();
   OOPLinkedListElement previous = new
            OOPLinkedListElement();
   if (head == null || head.value > x) {
            head = new OOPLinkedListElement();
            head.value = x;
            head.next = null;
   }
   else {
            current = head.next; previous = head;
            while (current != null && current.value < x) {
                     previous = current;
                     current = current.next;
            }
            a.value = x;
            a.next = current;
            previous.next = a;
   }
}

}
```

2.6 Exercise 19

```
package uk.ac.cam.tma33.s6;

public class OOPLazySortedLinkedList
                            extends OOPSortedLinkedList {
@ Override
public void Insert(int x)  {
   OOPLinkedListElement a = new OOPLinkedListElement();
   if (head!=null)
           { a.value = x; a.next = head; }
   else
           { a.value = x; a.next = null;}
   head = a;
}

@Override
public int getNth(int n) throws Exception {
   OOPSortedLinkedList sorted = new OOPSortedLinkedList();
   OOPLinkedListElement current = head;
   while (current != null) {
           sorted.Insert(current.value);
           current = current.next;
   }
   return sorted.getNth(n);
}

}
```

3 Polymorphism

3.1 Exercise 20

An abstract class is a class that can't be initialised. Its only purpose is for other classes to extend. Interfaces are very similar to abstract classes. It doesn't have any state whatsoever and all its methods are abstract. They allow multiple inheritance without any chance of conflict, thing which with normal classes would not have been possible.

3.2 Exercise 21

In OOP, *dynamic polymorphism* is crucial for good, clean code. In contrast to static polymorphism, where the method which is going to be used is determined at compilation, dynamic polymorphism determines it at runtime. Therefore, there also is a performance overhead associated to it. A good example in this case would be the Overriding of the Insert method from the OOPLinkedList class, used in Exercise 18 to create the OOPSortedLinkedList one.

3.3 Exercise 22

If we couldn't rely on the existence of an *instanceof* operator, the only possibility that we have left is polymorphism. In other words, we can consider *Shape* as being an instance rather than a class, and all the other classes (*Circle, Square*, etc.) will implement it. In this way, when we call *S.draw()*, where S is a shape, the compiler will look in the memory and will know exactly which class to refer to.

3.4 Exercise 23

In my opinion, the idea of *'selective inheritance'* would make our code run faster, because, in stead of using Overriding (i.e. dynamic polymorphism) to define how a certain method should act within a subclass.

3.5 Exercise 24

Student class

```
abstract public class Student {
        abstract public boolean pass();
}
```

CSStudent and *NSStudent* classes

```
public class CSStudent extends Student{
        private int ticks;
        public boolean pass(){
                if (ticks == 20) return true;
                else return false;
        }
}

public class NSStudent extends Student {
        private int ticks;
        public boolean pass() {
                if (ticks == 10) return true;
                else return false;
        }

}
```

3.6 Exercise 25

(a)
• *GetHead* —>O(1) In an array, we can access any element in constant complexity.
• *deleteHead* –>O(N). In order to delete the first element, we need to shift all the other elements left 1 position =>N-1 shifts =>O(N) complexity
• *Insert* –>O(N). In order to insert an element at the beginning of the array, we need to shift all the current elements 1 position to the right =>N shifts.
bullets length –>O(N). The algorithm has to go through all the elements in the array in order to count them.
bullets getNth –>O(1). Access in O(1) at any element

(b)

```
public interface OOPList {

        public int getHead() throws Exception;
        public void deleteHead() throws Exception;
        public void Insert(int x);
        public int length();
        public int getNth(int n) throws Exception;

}
```

(c)

OOPArrayList class :

```
public class OOPArrayList implements OOPList{
private int[] array;

public int getHead() throws Exception{
        if (length() == 0) throw new
                Exception("Empty list!");
        else return array[0];
}

public void deleteHead() throws Exception{
        if (length() == 0) throw new
                Exception("List already empty!");
        else {
                int N = length();
                int[] new_array = new int[N - 1];
                for (int i = 1; i<N; i++)
                        new_array[i-1] = array[i];
                array = new_array;
        }
}

public void Insert(int x){
        int N = length();
        int[] new_array = new int[N+1];
        for (int i = 1; i<=N; i++)
                new_array[i] = array[i-1];
        new_array[0] = x;
        array = new_array;
}

public int length() {
        return array.length;
}

public int getNth(int n) throws Exception{
int N = length();
        if (n>N) throw new
          Exception("There is no element at position" ...
           +Integer.toBinaryString(n));
        else return array[n-1];
}
}
```

3.7 Exercise 26

(a) The *Queue* interface :

```
public interface Queue {
        public int getHead();
        public void removeHead();
        public void insertValue(int x);
        public boolean empty();
}
```

(b) The methods used to reverse the function, from *OOPLinkedList* class :

```
private OOPLinkedListElement
                 reverse(OOPLinkedListElement current){
        OOPLinkedListElement Next = current.next;
        current.next = null;
        OOPLinkedListElement rev = reverse(Next);
        Next.next = current;
        return rev;
}

public void reverse(){
        if (head != null && head.next!=null)
                head = reverse(head);
}
```

OOPListQueue class :

```
public class OOPListQueue implements Queue{

private OOPLinkedList left, right;

private void swapLists(){
        right.reverse();
        left = right;
        right = new OOPLinkedList();
}

public int getHead() throws Exception{
        if (empty()) throw new Exception("Empty␣queue!");
        else {
                if (left.length() == 0) swapLists();
                return left.getHead();
        }
}

public void removeHead() throws Exception{
        if (empty()) throw new Exception("Empty␣queue!");
        else {
                if (left.length() == 0) swapLists();
                left.deleteHead();
        }
}

public void insertValue(int x) {
        right.Insert(x);
        if (left.length() == 0) swapLists();
}

public boolean empty(){
        if (left.length()==0 && right.length()==0)
                        return true;
        else return false;
}

}
```

OOPArrayQueue class :

```
public class OOPArrayQueue implements Queue{

private int[] Q;
private int left, right;

public int getHead()throws Exception{
        if (empty()) throw new Exception("Empty␣queue");
        else return Q[left];
}

public void removeHead()throws Exception{
        if (empty()) throw new Exception("Empty␣queue!");
        else left++;
}

public void insertValue(int x){
        if (right!=0) Q[++right] = x;
}

public boolean empty(){
        if (left>=right) return true;
        else return false;
}

}
```

(d)

(i) *OOPListQueue* :

- getHead = O(1);
- removeHead = O(1)
- insertValue = O(1) (but it is AMORTISED)
- empty = O(1);

(ii) *OOPArrayQueue* :

- getHead = O(1);
- removeHead = O(1)
- insertValue = O(1)
- empty = O(1);