Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Αντικειμενοστρεφής Προγραμματισμός
Βασίλης Χριστοφίδης

Πρόοδος (3 ώρες)
Ημερομηνία: 29 Νοεμβρίου 2008

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (9 μονάδες) // Χειρισμός Πινάκων

Σας δίνεται η παρακάτω μέθοδος Java:

```
public static void mystery(String [] arr) {
     for(int i=1; i<arr.length; i++) {
          arr[i] = arr[i-1] + arr[i];
     }
}
```

(α) **(2 μονάδες)** Ποιος είναι η μέση χρονική πολυπλοκότητα εκτέλεσης της μεθόδου `mystery()` σε σχέση με το μήκος του πίνακα εισόδου;

> **Λύση:**
> Είναι της τάξης του O(m), where m is the length of the array πιο συγκεκριμένα O(m-1)

(β) **(2 μονάδες)** Ποια είναι τα περιεχόμενα του πίνακα μετά την εκτέλεση του παρακάτω κώδικα;

```
String [] a = {"hwat?", "hwat?", "okay!"};
mystery(a);
```

> **Λύση:**
> value of `a`: `{"hwat?", "hwat?hwat?", "hwat?hwat?okay!"}`

Common Error
`value of a: {"hwat?", "hwat?hwat?", "hwat?okay!"}`

(γ) **(5 μονάδες)** Η κλήση της μεθόδου `mystery(a)` αντιγράφει τα περιεχόμενα της μεταβλητής `a` στην ουσιαστική παράμετρο `arr`. Στην συνέχεια η μέθοδος μεταβάλει περιεχόμενα της `arr`. Όταν ολοκληρωθεί η εκτέλεση της μεθόδου, τα περιεχόμενα της `a` έχουν επίσης μεταβληθεί. Εξηγήστε την αιτία αυτής της αλλαγής.

> **Λύση:**
> Arrays are references or pointers to the actual array elements. When the array a is copied to the array `arr`, all that is actually copied is the pointer to the elements, so that both `a` and `arr` then point to the same elements. Therefore, when the elements of `arr` are changed inside the method, that also changes the elements of `a`.

Common Error

Στην Java όταν περνάμε ορίσματα σε μεθόδους είναι πάντα call by value, όσοι έδωσαν λάθος απάντηση θεωρούσαν ότι είχαμε call by reference.

Άσκηση 2 (17 μονάδες) **// Χειρισμός Αρχείων και Συμβολοσειρών**

(α) **(4 μονάδες)** Υλοποιήστε μία μέθοδο reverseString(String) η οποία επιστρέφει ανεστραμμένη την συμβολοσειρά String που δίνεται σαν παράμετρος εισόδου.

Λύση:
```
public static String reverseString(String original) {
 String reverse = ""; // Start the reverse as an empty String
//loop through the characters of the original in reverse order
 for(int i=original.length()-1; i>=0; i--) {
     reverse = reverse + original.charAt(i);
//Alternative Solution: reverse + original.substring(i,i+1);
 }
 return reverse;
}
```

Common Errors

Πολλοί χρησιμοποιούσαν πίνακα char για να κάνουν τις μετατροπές και τον επέστρεφαν θεωρωντας πως και αυτός ήταν κάτι σαν String.
Χρήση StringBuffer κλάσης όπου και εκεί ορισμένοι την θεωρούσαν σαν String.
Χειρισμός Strings σαν να είναι char array –λανθασμένος τρόπος προσπέλασης.

(β) **(8 μονάδες)** Υλοποιήστε μία μέθοδο reverseLines(String, String) η οποία διαβάζει γραμμές από ένα αρχείο κειμένου (πρώτη παράμετρος) και γράφει γραμμές σε ένα αρχείο εξόδου (δεύτερη παράμετρος) που περιέχουν τους χαρακτήρες της γραμμής εισόδου σε αντίστροφη σειρά. Μπορείτε να χρησιμοποιήσετε την μέθοδο reverseString(String) του προηγουμένου ερωτήματος, υποθέτοντας ότι η υλοποίησή της είναι σωστή. Μπορείτε να χειριστείτε τις εξαιρέσεις στον κώδικά σας με όποιο τρόπο επιθυμείτε, αρκεί η μέθοδος να μπορεί να μεταγλωττιστεί με επιτυχία.

Λύση:
```
public static void reverseLines(String inputFile, String
outputFile) throws FileNotFoundException, IOException {
  BufferedReader br = new BufferedReader(new
                                        FileReader(inputFile))
;
  PrintWriter pw = new PrintWriter(new File(outputFile));
  String inline = br.readLine();
  while(inline!=null) {
     pw.println(reverseString(inline));
     inline = br.readLine();
 }
 br.close();
 pw.close();
}
```

Common Errors

Πολύ μπερδέμενος κώδικας σε ορισμένες περιπτώσεις.
Ανώφελη και λανθασμένη χρησιμοποίηση του StringTokenizer.Η χρήση αυτής της κλάσης ήταν λανθασμένη καθώς εμείς θέλαμε να κάνει reverse όλη τη γραμμή που διάβαζαν ενώ αυτοί έκαναν μόνο τις μεμονομένες λέξεις και της άφηναν στην ίδια σειρά

(γ) **(5 μονάδες)** Δεδομένου ότι στην Java, ένα `String` από την στιγμή που αρχικοποιείται δεν μπορεί με κανένα τρόπο να αλλάξει (*immutable*) εξηγήστε πως δουλεύει η παρακάτω μέθοδος:

```
public static void mystery(String[] arr) {
     String name = "Vassilis";
     name = name + " Christophides";
     System.out.println(name);
}
```

Λύση:
In the program, the variable name is bound to the String object whose value is "Vassilis", and in te assignment statement name = name + " Christophides"; Java will evaluate the right-hand side, creating a new String object, whose value is the concatenation of name + " Christophides". It will rebind the variable name to this new `String` object, and the previous `String` object that name was bound to becomes an orphan, that is, an object that no longer has any references o it and will be garbage collected.

Common errors
Λίγοι έγραψαν πλήρη και σωστή δικαιολόγηση.

Άσκηση 3 (20 μονάδες) **// Σύστημα Τύπων Java και Υπερφόρτωση Μεθόδων**

Σας δίνεται το παρακάτω πρόγραμμα Java:

```
import java.lang.*;
import java.util.*;
public class Quiz{
  public static void main(String args[]){
     probMethod(1,2);
  }
  public static void probMethod(int M, double N){
     System.out.println("M + N = " + (M + N));
  }
  public static void probMethod(double M, int N){
     System.out.println("M + N = " + (M + N));
  }
}
```

(α) (**10 μονάδες**) Γιατί το πρόγραμμα δεν μπορεί να μεταγλωττιστεί με επιτυχία;

Λύση:
The overloaded methods are fine. However, Java always tries to use overloading before it tries to use automatic type conversion. If Java can find a definition of a method that matches the types of the arguments, it will use that definition. Java will not do an automatic type conversion of a method's argument until after it has tried and failed to find a definition of the method with parameter types that exactly match the arguments in the method definition.
In this case, the methods differ in the type of their parameter, and so this is a valid overloading of the method name `probMethod`. However, Java cannot decide whether to convert the `int` value 2 to a `double` value 2.0 and use the first definition of `probMethod`, or whether to convert the `int` value 1 to a `double` value 1.0 and use the second definition.

(β) (**10 μονάδες**) Βρείτε και διορθώστε το λάθος του παραπάνω προγράμματος, χωρίς να διαγράψετε καμία υπερφορτωμένη μέθοδο. Τι θα τυπωθεί, μετά την διόρθωσή σας, στην έξοδο κατά την εκτέλεση της `main()`;

Λύση:
The error occurs in the method main, where the invocation of the method is made, since Java cannot decide which overloaded definition of `probMethod` to use. In this situation, Java issues an error message indicating that the method invocation `probMethod(1,2)` is ambiguous.
There are several ways to correct the error, the easiest solution is to change the method invocation to `probMethod(1.0,2)` or `probMethod(1,2.0)`. In both cases the output would be M + N = 3.0.
Another solution is to change the type of one of the parameters in one of the methods from double to int. Yet another solution is to write a separate method where both parameters are of type int. In all of these cases, the output would be M + N = 3.

Common Errors: Πολλοί φοιτητές στο ερώτημα (α) είπαν ότι το πρόβλημα ήταν το ότι οι μέθοδοι είχαν δηλωθεί static και δεν μπορούσαν να καλεστούν με αυτό το τρόπο από τη main ή ότι δεν υπήρχε η μέθοδος **probMethod**(int M, int N) για να καλεστεί. Ακόμα, κάποιοι είχαν απάντησει ότι το πρόβλημα ήταν ότι δεν μπορούσε να γίνει η πρόσθεση int και double ή ότι δεν μπορούσαν να υπάρχουν μέθοδοι με το ίδιο όνομα και τις ίδιες παραμέτρους. Στο ερώτημα (β) κάποιοι απάντησαν ότι η λύση θα ήταν η διαγραφή ή η μετονομασία της μίας συνάρτησης ή ότι έπρεπε να γίνεται type casting στην πρόσθεση μέσα στην println().

Άσκηση 4 (15 μονάδες) **// Πολυμορφισμός και η δήλωση final**

Σας δίνονται οι παρακάτω δύο κλάσεις Java:

```
public class BadBase {
	private int x;
	// handle simple initializations
	protected void init() {
		x = 10;
	}
	public int getX() { return x; }
	public BadBase() { init(); }
}
public class BadSubclass extends BadBase {
	private int y;
	protected void init() {
		y = 20;
	}
	public int getY() { return y; }
	public BadSubclass() { init(); }
	public static void main(String args[]) {
		BadSubclass bsc = new BadSubclass();
		System.out.println("x = " + bsc.getX());
		System.out.println("y = " + bsc.getY());
	}
}
```

(α) **(2 μονάδες)** Τι τυπώνεται στην έξοδο του προγράμματος `main()` της `BadSubclass`;

Λύση:
x = 0
y = 20

(β) **(5 μονάδες)** Εξηγήστε το βασικό σχεδιαστικό λάθος των παραπάνω κλάσεων, δηλ. ποια σχεδιαστική αρχή του αντικειμενοστρεφούς προγραμματισμού παραβιάζουν;

Λύση:

In general, if a method in a base class calls another method which is not final, then the behavior of subclass methods depends on the base class implementation. This is a violation of the *principle of encapsulation*. In particular, a constructor should never call any method which is not final. The `BadBase` constructor calls init to initialize x, but `BadSubclass` overrides init so the base class variable x is never initialized and retains its default initialization value of 0.

Common Error: Κάποιοι φοιτητές δεν έδιναν πλήρη απάντηση ή έγραφαν περιττές επεξηγήσεις. Πάντως οι περισσότεροι είχαν πιάσει τη βασική ιδέα απλά δεν ήξεραν να το εξηγήσουν πλήρως.

(γ) **(8 μονάδες)** Στην Java, η δεσμευμένη λέξη **`final`** έχει διαφορετικές σημασίες ανάλογα με το πλαίσιο που χρησιμοποιείται. Μια μεταβλητή που δηλώνεται σαν **`final`** είναι μια σταθερά: η τιμή της δεν μπορεί να αλλάξει μετά την αρχικοποίησή της. Μια μέθοδος δηλώνεται σαν **`final`** δεν μπορεί να έχει υποσκελισμένη υλοποίηση σε υποκλάσεις. Μια κλάση που δηλώνεται σαν **`final`** δεν μπορεί να εξειδικευτεί σε υποκλάσεις. Εξηγήστε γιατί είναι πραγματικά χρήσιμο να δηλώνουμε μια κλάση ή μέθοδο σαν **`final`**; Δώστε την πληρέστερη δυνατή αιτιολόγηση (εξηγήστε τα επιχειρήματά σας σε ένα παράδειγμα).

Λύση:
Polymorphism is a useful feature, but it can cause problems:

```
void f(SomeClass arg) {
 arg.someMethod(); // what function is called here?
...
}
```

We do not know what function is being called in the expression `arg.someMethod()`. When the client calls `f`, he may provide as `arg` an instance of `SomeClass`, or he may substitute an instance of a subclass of `SomeClass`. If the subclass overrides the method `someMethod`, then the expression calls the subclass's method. If, however, `SomeClass` is `final`, or `SomeClass.someMethod` is `final`, then we know for certain that the expression calls the method `SomeClass.someMethod`.
When designing a framework, you create a class that is intended to be used as the base class for inheritance. It is important to provide documentation for implementers of subclasses. You should document which functions subclasses are expected to implement or override. If a subclass is not supposed to override a particular function, you can make this clear by declaring that method final.
We did not discuss this, but code for calling final method is also more efficient. The function call can be bound at compile-time, i.e., the compiler can generate a call to the function instead of calling the runtime dispatcher.

Common Error: Επειδή είναι περισσότερο θεωρητική η ερώτηση αυτή, το κοινό λάθος ήταν να γράφουν μη πλήρη απάντηση η αρκετά περιττά. Πάλι όμως οι περισσότεροι είχαν πιάσει τη γενική ιδέα.

Άσκηση 5 (12 μονάδες) // **κληρονομικότητα**

Σας δίνονται οι παρακάτω κλάσεις Java:

```
public class Ice extends Fire {
 public void method1() {
   System.out.println("Ice 1");
 }
}
public class Rain extends Fire {
 public String toString() {
  return "Fire";
 }
 public void method1() {
  System.out.println("Fire 1");
 }
 public void method2() {
  System.out.println("Fire 2");
 }
}
public class Snow extends Rain {
 public void method2() {
  System.out.println("Snow 2");
 }
}
```

```
  return "Rain";
 }
 public void method1() {
  System.out.println("Rain 1");
 }
}
```

Τι τυπώνεται στην έξοδο του ακόλουθου προγράμματος Java;

```
Fire[] elements = {new Fire(), new Snow(), new Rain(), new Ice()};
for (int i = 0; i < elements.length; i++) {
     System.out.println(elements[i]);
     elements[i].method1();
     elements[i].method2();
     System.out.println();
    }
```

Λύση:

```
Fire
Fire 1
Fire 2
Rain
Rain 1
Snow 2
Rain
Rain 1
Fire 2
Fire
Ice 1
Fire 2
```

Common Error: Μερικοί φοιτητές δεν εκτύπωναν τίποτα ή εκτύπωναν null στην εντολή System.out.println(elements[i]). Μάλλον δεν αναγνώρισαν ότι καλείται έμμεσα η μέθοδος toString() των αντικειμένων που βρίσκονται στον πίνακα.

Άσκηση 6 (10 μονάδες) // Χειρισμός Εξαιρέσεων

Σας δίνεται ο παρακάτω κώδικας Java:

```
try {
  System.out.println(1);
  try {
    System.out.println(2);
    f ();
    System.out.println(3);
  }
  catch (E1 e1) {
    System.out.println(4);
  }
  finally {
    System.out.println(5);
  }
 }
 catch (E2 e2) {
  System.out.println(6);
 }
```

Τι θα τυπωθεί στην έξοδο αυτού του προγράμματος στην περίπτωση που (α) δεν εγείρονται εξαιρέσεις (exception), (β) η f προκαλεί (throws) μια εξαίρεση τύπου E1, (γ) η f προκαλεί μια εξαίρεση τύπου E2, και (δ) η f προκαλεί μια εξαίρεση τύπου E3.

Λύση:
(α) 1, 2, 3, 5
(β) 1, 2, 4, 5
(γ) 1, 2, 5, 6
(δ) 1, 2, 5, **Exception**

Common Errors: Οι περισσότεροι φοιτητές δεν εκτύπωναν στο ερώτημα (δ) ότι θα «εκτυπωθεί» και το exception (αφού δεν γίνεται catch). Μερικοί δεν αναγνώρισαν ότι ο κώδικας σε ένα finally block εκτελείται(ακόμα και αν δεν προκληθεί εξαίρεση στο αντίστοιχο μπλοκ try), οπότε και δεν εκτύπωναν τον αριθμό 5.

Άσκηση 7 (24 μονάδες) // Αφαιρετικοί Τύποι Δεδομένων

Σχεδιάστε και υλοποιήστε έναν αφαιρετικό τύπο δεδομένων (ΑΤΔ) Μιγαδικός Αριθμός (Complex Number). Ένας μιγαδικός αριθμός αποτελείτε από ένα πραγματικό (**real**) και ένα φανταστικό (**imaginary**) μέρος. Θεωρούμε ένα συμβόλαιο για τον ΑΤΔ που περιλαμβάνει τις εξής λειτουργίες:

- Δημιουργία ενός νέου μιγαδικού αριθμού
- Επιστροφή του πραγματικού μέρους
- Επιστροφή του φανταστικού μέρους
- Έλεγχος εάν δύο μιγαδικοί αριθμοί είναι ίσοι
- Πρόσθεση δύο μιγαδικών αριθμών
- Αφαίρεση δύο μιγαδικών αριθμών
- Επιστροφή του συζυγή ενός μιγαδικού αριθμού
- Επιστροφή μιας αλφαριθμητικής αναπαράστασης του μιγαδικού αριθμού

Σημασιολογία:

- Ένας μιγαδικός αριθμός αρχικοποιείται κατά την δημιουργία του περνώντας ως παραμέτρους το πραγματικό και φανταστικό του μέρος: **(a,b)** όπου το **a** και το **b** δεν είναι συγχρόνως μηδέν.
- Το πραγματικό και το φανταστικό μέρος ενός μιγαδικού είναι και οι δύο πραγματικοί αριθμοί (`float`). Το φανταστικό μέρος ακολουθεί ο φανταστικός αριθμός **i**, που έχει την ιδιότητα $\mathbf{i^2 = -1}$: **x = a + ib**.
- Για να είναι ένας μιγαδικός αριθμός ίσος με έναν άλλον, θα πρέπει τα πραγματικά και τα φανταστικά τους μέρη να είναι αντίστοιχα ίσα.
- Για να προσθέσουμε δύο μιγαδικούς αριθμούς, προσθέτουμε πρώτα τα πραγματικά τους μέρη και μετά τα φανταστικά: **(a + bi) + (c + di) = (a+c) + (b+d)i**
- Για να αφαιρέσουμε δύο μιγαδικούς αριθμούς, αφαιρούμε πρώτα τα πραγματικά τους μέρη και μετά τα φανταστικά: **(a + bi) − (c + di) = (a−c) + (b−d)i**
- Για να βρούμε τον συζυγή ενός μιγαδικού αριθμού παίρνουμε τον κατοπτρισμό του ως προς τον άξονα των πραγματικών: $\bar{z}$ **= a - ib**
- Η αλφαριθμητική αναπαράσταση πρέπει να είναι της μορφής: **a + b * i**, όπου **a** είναι το πραγματικό μέρος και **b** το φανταστικό.

(α) **(8 μονάδες)** Καθορίστε την προδιαγραφή του ΑΤΔ Μιγαδικός Αριθμός, δηλαδή δώστε τις υπογραφές (signatures) των προβλεπόμενων λειτουργιών του, το είδος τους ανάλογα με την επίδραση που έχουν στην κατάσταση των αντικειμένων (constructors, observers, accessors, applicative/mutative transformers), καθώς και τις εκ των προτέρων, τις εκ των υστέρων και τις αμετάβλητες συνθήκες (preconditions, postconditions, invariants).

Λύση: type complex **imports** real
operations:
//Constructor

```
        init: float X float → complex
//Transformers Applicative
        +:complex X complex → complex //addition
        -: complex X complex → complex //subtraction
//      *: complex X complex → complex //multiplication
//      /: complex X complex → complex //division
        -: complex → complex // conjugate
//Accesors
        getReal: complex → real
        getImaginary: complex → real
         equals: Complex X Object → boolean
         tostring: complex → String
variables: x,y,z: complex; a,b: real
axioms:
         getReal(init(a,b)) = a
        getImaginary(init(a,b)) = b
        getReal(x+y) = getReal(x) + getReal(y)
        getImaginary(x+y) = getImaginary(x) + getImaginary(y)
         getReal(x*y) = getReal(x)*getReal(y)-getImaginary(x)*imag(y)
         getImaginary(x*y) = getReal(x)*getImaginary(y)- getImaginary (x)*reapartl(y)
         getReal(conjugate (x)) = getReal(x)
        getImaginary(conjugate(x)) = - getImaginary(x)
         conjugate (conjugate (x)) = x
         equals(x,y) ⇔getReal(x)=getReal(y) and getImaginary(x)=getImaginary(y))
invariants: for all x: complex !(getReal(x) =0 && getImaginary(x) =0)
violations:
         init(a,b) requires  !(a=0 && b=0)
         +(x,y) requires !(getReal(x) + getReal(y)=0 &&
                          getImaginary(x) + getImaginary(y)=0)
         -(x,y) requires !(getReal(x) - getReal(y)=0 &&
                          getImaginary(x) - getImaginary(y)=0)
```

(β) **(16 μονάδες)** Υλοποιήστε μία κλάση στιγμιοτύπων η οποία σέβεται το συμβόλαιο του ΑΤΔ Μιγαδικός Αριθμός που δώσατε στο προηγούμενο ερώτημα. Στην υλοποίησή σας υποσκελίστε (override) την υλοποίηση των μεθόδων `equals()` και `toString()` που υποστηρίζει η κλάση `Object`. Ελέγξετε την ορθότητα των ισχυρισμών (assertions) του συμβολαίου (post/invariant conditions) κατά τη διάρκεια εκτέλεσης του κώδικά σας. Χρησιμοποιήστε εξαιρέσεις για παραβιάσεις των προ συνθηκών (pre conditions).

Λύση:

```
/**
 * This class represents an Abstract Data Type of a Complex
  * number. A Complex consists of a real and an imaginary
part,
 * called Cartesian coordinates.
 * @author Vassilis Christophides
 * @version 1.0
*/
public class Complex {
    private float re, im;
//** Constructor **//
    /**
     * Creates a complex number out of a real&imaginary value
     * precondition: init(a,b) requires  !(a=0 && b=0)
     * postcondition: getReal(init(a,b)) = a,
      * getImaginary(init(a,b)) = b
     * @param re the real part of the complex number
```

```
      * @param im the imaginary part of the complex number
       * @throws IllegalArgumentException
      * @return a non null object of this class
      */
     public Complex(float re, float im) {
        if (re != 0) || (im != 0) {
          this.re = re;
          this.im = im;
          }
          else  throw  new  IllegalArgumentException("Non  zero
args");
     }

//** (Applicative) Transformers **//
     /**
      * Returns a Complex whose value is this (x) + other (y).
       * precondition: getReal(x+y)!=0 or getImaginary(x+y)!=0
      * postcondition: getReal(x+y) = getReal(x) + getReal(y)
      * getImaginary(x+y) = getImaginary(x) - getImaginary(y)
      * @param other Complex to be added to this Complex
      * @return a non null object of this class
      *
      */
     public Complex add(Complex other) {
          if(re+other.getReal() == 0 &&
              im+other.getImaginary == 0)
             throw new IllegalArgumentException
             ("You can't have as a result of the addition a
    Complex  with  both  real  and  imaginary  parts  equal  to
zero.");
          Complex sum = new Complex(re + other.getReal(),
                                    im + other.getImaginary());
          assert (sum.getReal() == re + other.getReal()) &&
                                  (sum.getImaginary()  ==  im  +
other.getImaginary());
          return sum;
     }
     /**
      * Returns a Complex whose value is this (x) - other (y).
      * precondition: getReal(x-y)!=0 or getImaginary(x-y)!=0
      * postcondition: getReal(x-y) = getReal(x) - getReal(y),
      * getImaginary(x-y) = getImaginary(x) - getImaginary(y)
      * @throws IllegalArgumentException
      * @param other Complex to be subtracted to this Complex
      * @return a non null object of this class
      */
     public Complex sub(Complex other) {
          if(re-other.getReal() == 0 &&
              im-other.getImaginary == 0)
             throw new IllegalArgumentException
             ("You can't have as a result of the subtraction a
    Complex  with  both  real  and  imaginary  parts  equal  to
zero.");
          Complex sub = new Complex(re - other.getReal(),
                                    im - other. getImaginary());
          assert (sub.getReal() == re - other.getReal()) &&
                                  (sub.getImaginary()  ==  im  -
other.getImaginary());
          return sub;

     }
     /**
      * Returns a Complex whose value is conjugate of this (x)
```

```
     * precondition: none
     * postcondition: getReal(conjugate (x)) = getReal(x),
      * getImaginary(conjugate(x)) = - getImaginary(x)
     * @return a non null object of this class
     */
    public Complex conj() {
         Complex conj = new Complex(re, -im)
      assert(conj.getReal()==re)&&(conj.getImaginary()==-im);
         return conj;
    }

//** Accessors **//
    /**
     * Returns the real part of this complex number.
     * precondition: none
     * postcondition: getReal(init(a,b)) = a
     * @return the value of the real part
     */
    public float getReal() {
         return re;
    }
    /**
     * Returns the imaginary part of this complex number.
     * precondition: none
     * postcondition: getImaginary(init(a,b)) = b
     * @return the value of the imaginary part
     */
    public float getImaginary() {
         return im;
    }
    /**
     * Returns a string representation of this complex number
     * precondition: none
     * postcondition: a string is created to represent the
     * data of this Complex in the form of
      * getReal() + getImaginary *i
     * @return a string representing this complex number
     */
    public String toString() {
         return re + " + " + im + "i";
    }
//** Observers **//
    /**
     * Compares this (x) complex value with the other (y).
     * precondition: none
     * postcondition: equals(x,y) ⇔getReal(x)=getReal(y) &&
      * getImaginary(x)=getImaginary(y))
     * @param other Complex to be compared with this Complex
     * @return true if this complex is equal to other, false
     * if object is null, not an instance of Complex, or
      * not equal to this Complex instance
     */
    public boolean equals(Object other) {
      boolean ret;
      if (this == other) {
          ret = true;
      } else if (other == null) {
          ret = false;
      } else  {
         try {
           Complex cother = (Complex)other;
           ret = (re==cother.getReal() &&
                  im== cother.getImaginary())
```

```
          } catch (ClassCastException ex) {// ignore exception
            ret = false;
          }
        }
      return ret;
    }
}
```

Common Errors:

- Αρκετοί φοιτητές ενώ είχαν υλοποιήσει σωστα την equals, δεν έβαζαν σαν όρισμα Object για να κάνουν override την equals που υποστηρίζει η κλάση Object.
- Δεν χρησιμοποιούσαν assert για να ελέγξουν ότι τα αποτελέσματα των πράξεων που έκαναν στις add,sub,conjurate είναι τα αναμενόμενα με βάση τα post conditions.
- Δεν έλεγχαν αν ο νέος Complex μετά τις πράξεις add, sub είναι ένας έγκυρος Complex και απλά έθεταν το φανταστικό και πραγματικό του μέρος ίσο με το αποτέλεσμα των πράξεων.
- Υπήραχαν κάποιες ασυνέπειες μεταξύ συμβολαίου και υλοποίησης. Π.χ. ανέφεραν τα preconditions στο συμβόλαιο αλλα δεν τα υποστήριζαν στην υλοποίηση και το αντίστροφο.
- Τέλος κάποιοι δεν ανέφεραν τον τύπο των μεθόδων (accessors/transformers/constructor/observer) ή τα έγραφαν λάθος. Οι περισσότεροι όμως τα είχαν σωστά.