
Καφέζας Γιώργος
Α.Μ. 4465



Εργαστήριο Οντοκεντρικού Προγραμματισμού I (JAVA)

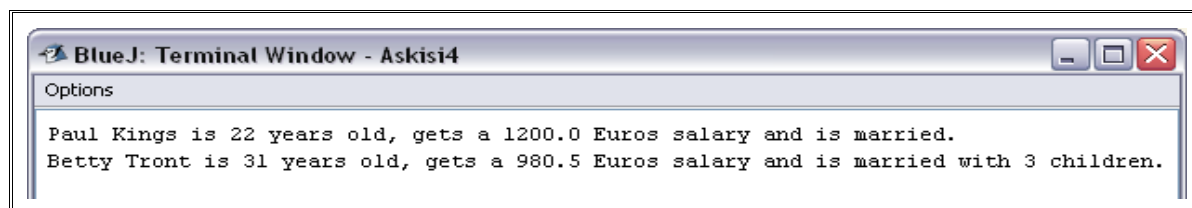
Ακαδημαϊκό Έτος 2011-2012

2^ο Σετ Ασκήσεων

4^η Εργαστηριακή Άσκηση

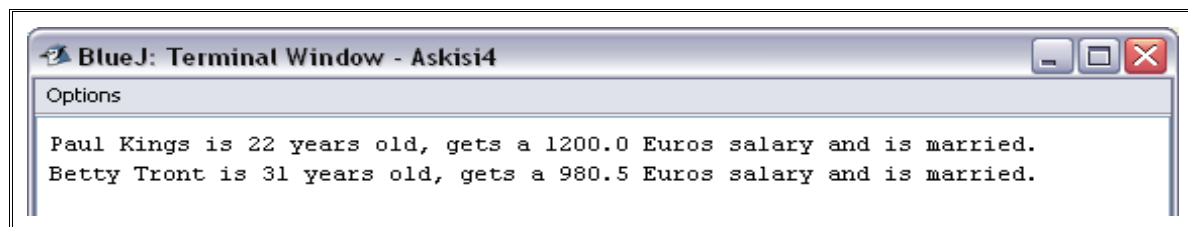
1) Γνωρίζουμε ότι μια κλάση που ορίζεται ως `abstract` δεν μπορεί να έχει στιγμιότυπα, ότι ενδέχεται να περιλαμβάνει μία ή περισσότερες `abstract` μεθόδους. Συνήθως ορίζουμε έτσι μια κλάση όταν ξέρουμε εξ αρχής πως δε θα δημιουργήσει στιγμιότυπα ή και όταν θέλουμε οι υποκλάσεις της να καθορίσουν τη δομή και την συμπεριφορά της. Τέτοια δήλωση γίνεται συνήθως σε κλάση που είναι στο ανώτερο επίπεδο ενός δέντρου κληρονομικότητας. Έτσι, ορίσαμε την κλάση `MyTester` ως `abstract` διότι είναι μια κλάση η οποία δεν θα έχει στιγμιότυπα, μιας και περιλαμβάνει την `main`. Δεν ήταν απαραίτητο, όμως σύμφωνα με τα παραπάνω, καταλαβαίνουμε ότι υπάρχει κάποιο όφελος.

Εκτελώντας τη `MyTester`, τώρα, παίρνουμε τα ακόλουθα αποτελέσματα:



Παρατηρούμε ότι εκτυπώνονται τα στοιχεία των δύο στιγμιότυπων `p1` και `mp1`, των κλάσεων `Person` και `MarriedPerson` αντίστοιχα.

2) Το βασικό πλεονέκτημα του δοθέντος κώδικα σε σχέση με τον προηγούμενο είναι πως για την εκτύπωση των στοιχείων καλείται η μέθοδος `printInfo()` έναντι των πολλών γραμμών κώδικα που απαιτούνταν πριν. Τρέχοντας το πρόγραμμά μας έχουμε τα εξής αποτελέσματα:



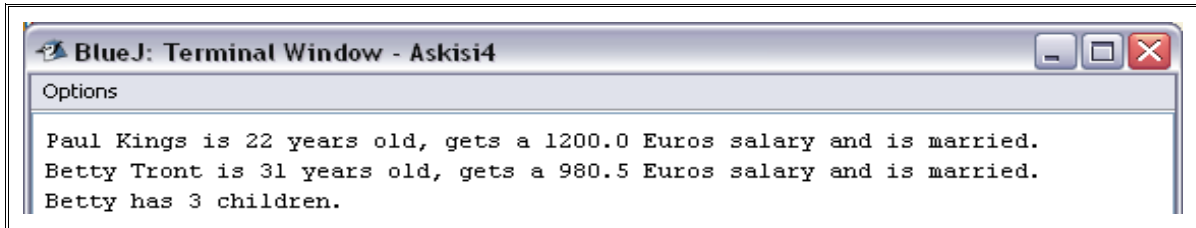
Παρατηρούμε ότι ξανά εκτυπώνονται τα στοιχεία των στιγμιότυπων `p1` και `mp1`, με τη διαφορά, όμως, ότι για το δεύτερο στιγμιότυπο δεν εκτυπώνεται ο αριθμός των παιδιών, δηλαδή η τιμή της μεταβλητής `children` και το ανάλογο κείμενο «...with _ children.». Κι αυτό συμβαίνει διότι η μέθοδος `printInfo()` βρίσκεται στην κλάση `Person` και συνεπώς δεν μπορεί να περιλαμβάνει εκτύπωση της μεταβλητής που βρίσκεται στην υποκλάση `MarriedPerson`.

3) Το παραπάνω πρόβλημα μπορεί να διορθωθεί γράφοντας μια μέθοδο `printInfo()` για την κλάση `MarriedPerson`, η οποία θα υπερκαλύπτει αυτήν της `Person`. Κάνοντας χρήση της `super`, έχουμε τον ακόλουθο κώδικα:

```
public void printInfo() {
    super.printInfo();
    System.out.print(this.getFirstName() + " has ");
    if (this.getNoOfChildren() > 0)
        System.out.print(this.getNoOfChildren());
    else System.out.print("no");
    System.out.println(" children.");
}
```

}

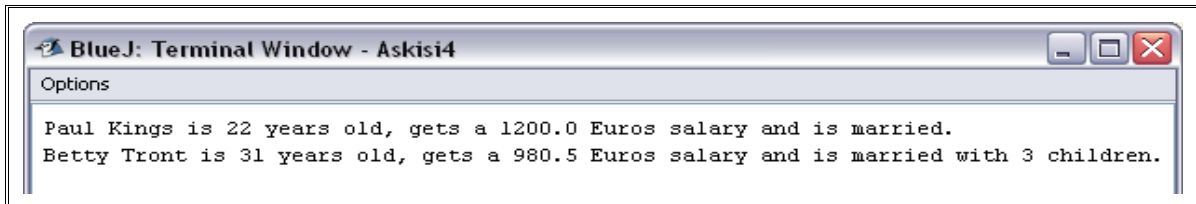
Τα αποτελέσματα έπειτα από την εκτέλεση του προγράμματος είναι τα εξής:



Με τον παραπάνω τρόπο βλέπουμε πως ναι μεν εκτυπώνονται όλες οι πληροφορίες και για τα στιγμιότυπα της `MarriedPerson`, αλλά εκτυπώνονται με τη μορφή δύο προτάσεων. Έτσι, θα μπορούσαμε να χρησιμοποιήσουμε εναλλακτικά την παρακάτω μέθοδο, ώστε η αρχική μορφή της εκτύπωσης των πληροφοριών να διατηρηθεί:

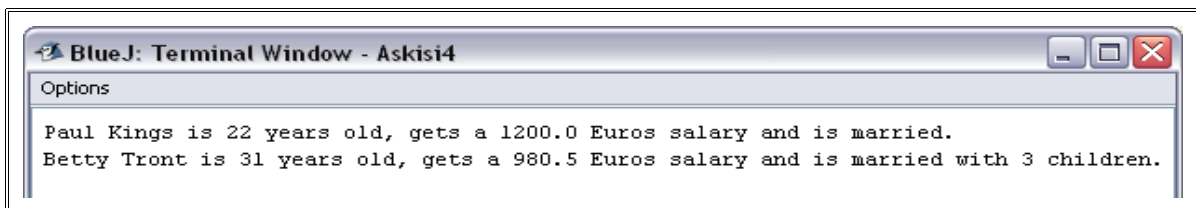
```
public void printInfo() {  
    System.out.print(this.getFirstName()+" "+this.getLastName()  
        +" is " +this.getAge()+ " years old, gets a "  
        +this.getSalary()+" Euros salary and is" + " married with ");  
    if (this.getNoOfChildren() > 0)  
        System.out.print(this.getNoOfChildren());  
    else System.out.print("no");  
    System.out.println(" children.");  
}
```

Σε αυτή την περίπτωση, τα αποτελέσματα θα ήταν τα εξής:



Παρατηρούμε ότι τώρα όντως τα αποτελέσματα είναι όμοια με αυτά στα πρώτα ερωτήματα της άσκησης αυτής. Δεν ξεχάσαμε να την μεταφράσουμε πάλι, άλλωστε είναι δυνατόν να έχουμε τα ίδια αποτελέσματα με χρήση διαφορετικού κώδικα και μεθόδων. Παρόλ' αυτά, στη συνέχεια του προγράμματος θα χρησιμοποιήσουμε την πρώτη μέθοδο.

4) δ. Έπειτα από τις αλλαγές και την εκτέλεση του προγράμματος, παρατηρούμε ότι τα αποτελέσματα είναι ίδια με πριν:



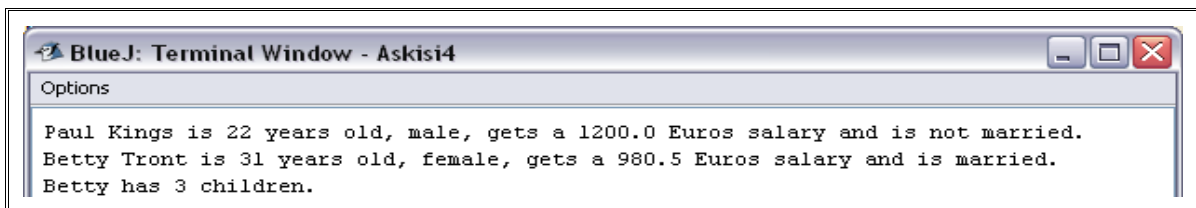
ε. Για να διορθώσουμε το παραπάνω, δηλαδή να εκτυπώνεται και η τιμή της μεταβλητής `sex` για κάθε στιγμιότυπο και σύμφωνα με την υπόδειξη της άσκησης, τροποποιούμε τη μέθοδο `printInfo()` στην κλάση `Person` (και κατ' επέκταση θα αλλάξει και στην `MarriedPerson`) ως κάτωθι:

```

public void printInfo() {
    System.out.print(this.getFirstName()+" "+this.getLastName()+" is "
        +this.getAge()+" years old, ");
    if(sex==0) System.out.print("male, ");
    else if (sex==1) System.out.print("female, ");
    System.out.print("gets a "+this.getSalary()+
        " Euros salary and is");
    if (this.isMarried() == false)
        System.out.print(" not");
    System.out.println(" married.");
}

```

Έτσι, μετά από την παραπάνω μετατροπή, έχουμε τα ακόλουθα αποτελέσματα:



```

BlueJ: Terminal Window - Askisi4
Options
Paul Kings is 22 years old, male, gets a 1200.0 Euros salary and is not married.
Betty Tront is 31 years old, female, gets a 980.5 Euros salary and is married.
Betty has 3 children.

```

5) α. Αρχικά, ορίζουμε μια μέθοδο στην κλάση `Person` για να μπορεί να μας επιστρέφει την τιμή της μεταβλητής `sex`:

```

public int getSex() { return sex; }

```

Έπειτα, ορίζουμε τις μεθόδους που μας υποδεικνύονται από την εκφώνηση:

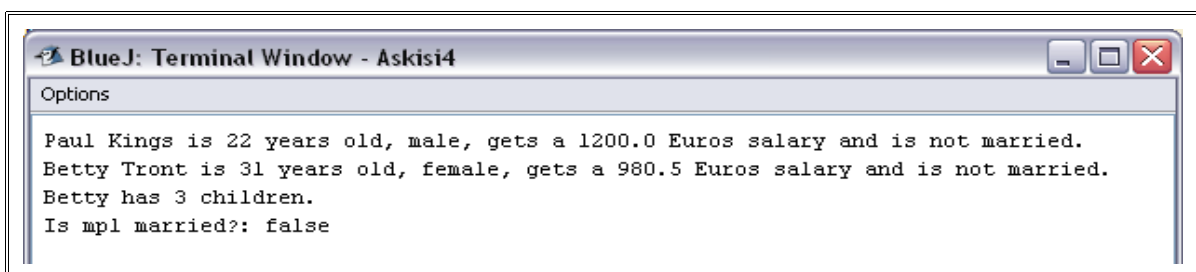
```

public void setLastName(String ln) { this.lastname=ln; } //Person
public void setFirstName(String fn) { this.firstname=fn; }
public void setAge(int a) { this.age=a; }
public void setMarried(boolean m) { this.married=m; }
public void setSalary(float s) { this.salary=s; }
public void setSex(int s) { this.sex=s; }

public void setNoOfChildren(int c) {this.children=c;} //MarriedPerson

```

β. Αλλάζοντας τον κώδικα και τρέχοντας το πρόγραμμα, παίρνουμε τα παρακάτω αποτελέσματα:



```

BlueJ: Terminal Window - Askisi4
Options
Paul Kings is 22 years old, male, gets a 1200.0 Euros salary and is not married.
Betty Tront is 31 years old, female, gets a 980.5 Euros salary and is not married.
Betty has 3 children.
Is mpl married?: false

```

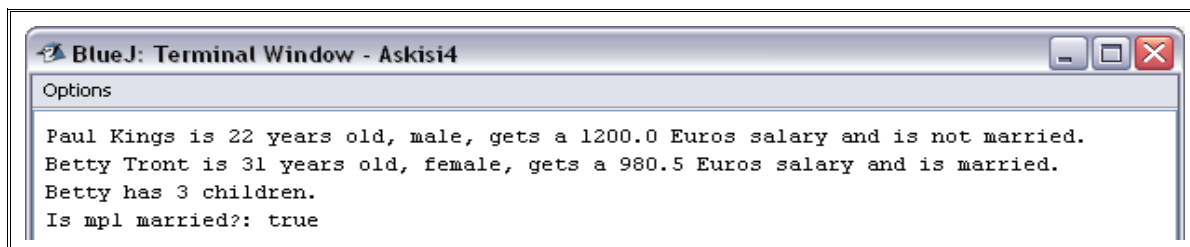
Παρατηρούμε ότι και για το στιγμιότυπο `mpl` της κλάσης `MarriedPerson` αλλάζει η μεταβλητή `married`, ενώ δεν θα έπρεπε, μιας και τα στιγμιότυπα αυτής της κλάσης εκ δημιουργίας τους και εκ φύσεως πρέπει να έχουνε `true` τιμή στη μεταβλητή `married`.

γ. Μπορούμε να διορθώσουμε την παραπάνω λανθασμένη λειτουργία υπερκαλύπτοντας τη

μέθοδος `setMarried()` στην κλάση `MarriedPerson` ως εξής:

```
public void setMarried(boolean m) { }
```

Ουσιαστικά, μόλις καλείται από στιγμιότυπο της `MarriedPerson`, επειδή δεν έχει καμία εντολή προς εκτέλεση στο σώμα της, απλά δε θα γίνεται τίποτα, και κατά συνέπεια, δε θα αλλάζει η τιμή της μεταβλητής `married`. Προς επιβεβαίωση της ορθής πλέον λειτουργίας, τρέχουμε ξανά το πρόγραμμα και παίρνουμε τα παρακάτω αποτελέσματα:

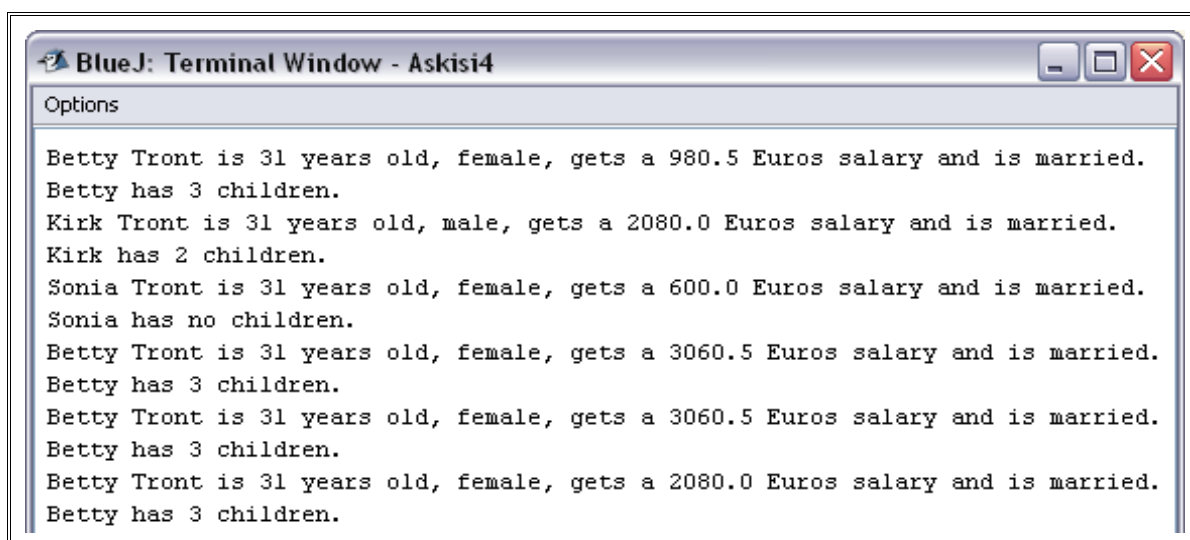


```
BlueJ: Terminal Window - Askisi4
Options
Paul Kings is 22 years old, male, gets a 1200.0 Euros salary and is not married.
Betty Tront is 31 years old, female, gets a 980.5 Euros salary and is married.
Betty has 3 children.
Is mp1 married?: true
```

6) Δημιουργώντας μια μέθοδο `setSalary()` όπως μας υποδεικνύεται στην εκφώνηση, έχουμε:

```
public void setSalary(MarriedPerson spouse) {
    float temp_sal;
    if (spouse.getSex() != this.getSex()) {
        temp_sal = this.getSalary() + spouse.getSalary();
        this.setSalary(temp_sal);
    }
}
```

Αντικαθιστώντας και τον κώδικα στην `MyTester` και τρέχοντας πάλι το πρόγραμμα, έχουμε:



```
BlueJ: Terminal Window - Askisi4
Options
Betty Tront is 31 years old, female, gets a 980.5 Euros salary and is married.
Betty has 3 children.
Kirk Tront is 31 years old, male, gets a 2080.0 Euros salary and is married.
Kirk has 2 children.
Sonia Tront is 31 years old, female, gets a 600.0 Euros salary and is married.
Sonia has no children.
Betty Tront is 31 years old, female, gets a 3060.5 Euros salary and is married.
Betty has 3 children.
Betty Tront is 31 years old, female, gets a 3060.5 Euros salary and is married.
Betty has 3 children.
Betty Tront is 31 years old, female, gets a 2080.0 Euros salary and is married.
Betty has 3 children.
```

Παρατηρούμε ότι εκτυπώνονται αρχικά τα στοιχεία των τριών στιγμιότυπων. Στη συνέχεια καλείται 3 φορές η μέθοδος `setSalary()` από το στιγμιότυπο `mp1`, με διαφορετικά ορίσματα κάθε φορά. Γενικά διαπιστώνουμε πως ανάλογα με το όρισμα που δίνεται στη μέθοδο, καλείται και διαφορετική, είτε από την υποκλάση `MarriedPerson` είτε από την υπερκλάση `Person`.

Πιο αναλυτικά, στην πρώτη κλήση της είναι `"mp1.setSalary(mp2);"`, όπου συγκρίνεται το φύλο των `mp1` και `mp2`, δεν είναι ίδιο και έτσι αλλάζει η τιμή του μισθού, σύμφωνα με την εκφώνηση. Στη δεύτερη κλήση, `"mp1.setSalary(mp3);"`, συγκρίνεται το φύλο, είναι ίδιο κι έτσι δεν αλλάζει τίποτα.

Τέλος, στην τρίτη κλήση, “mp1.setSalary(mp2.getSalary());” παρατηρούμε ότι το όρισμα είναι ένας float, συνεπώς καλείται η μέθοδος setSalary() που βρίσκεται στο σώμα της κλάσης Person, γι' αυτό και η μεταβλητή salary του mp1 παίρνει την τιμή της salary του mp2.

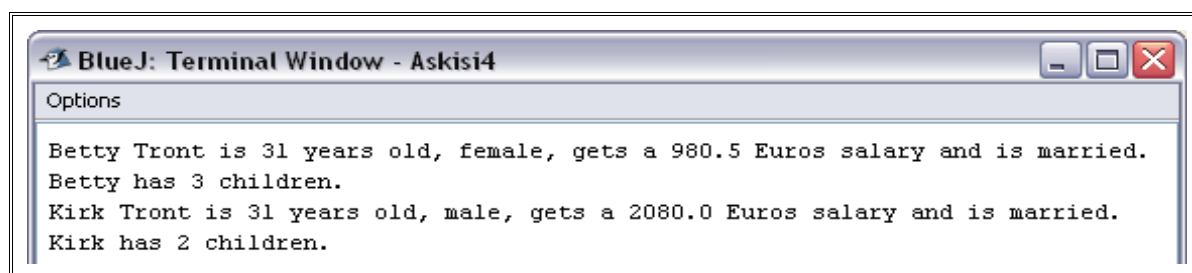
7) α. Για να δημιουργήσουμε ένα πίνακα προσθέτουμε τον ακόλουθο κώδικα στο σώμα της main:

```
MarriedPerson mpArray[]= new MarriedPerson[2];  
mpArray[0]=mp1;  
mpArray[1]=mp2;
```

Δυστυχώς δε μπορούμε να προσθέσουμε και το τρίτο στιγμιότυπο, γιατί κατά την δημιουργία ενός πίνακα, όπως ο παραπάνω, πρέπει να ορίζεται το μήκος του. Για να προσπελάσουμε τα στοιχεία του πίνακα και να εκτυπώσουμε τα αντίστοιχα δεδομένα, προσθέτουμε τον παρακάτω κώδικα:

```
for(int i=0 ; i<2 ; i++){  
    mpArray[i].printInfo();  
}
```

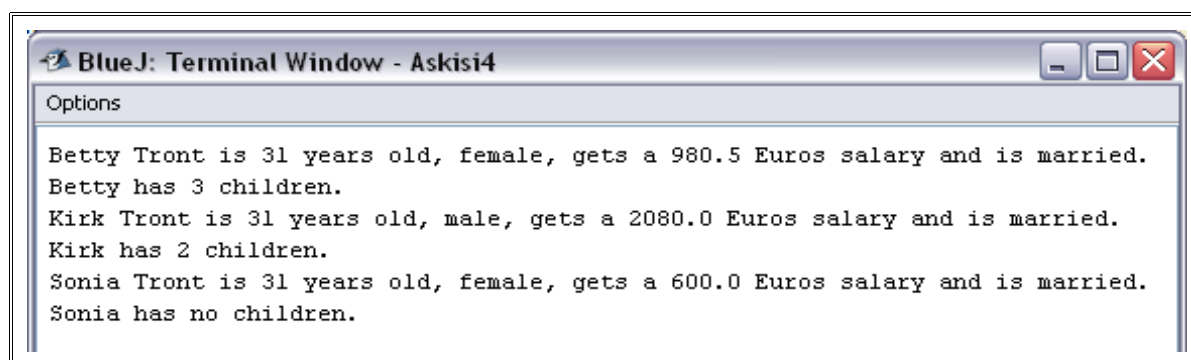
Τα αποτελέσματα έπειτα από την εκτέλεση του προγράμματος είναι:



β. Για να δημιουργήσουμε μια λίστα με τα τρία στιγμιότυπα που έχουμε, προσθέτουμε τον ακόλουθο κώδικα στο σώμα της main:

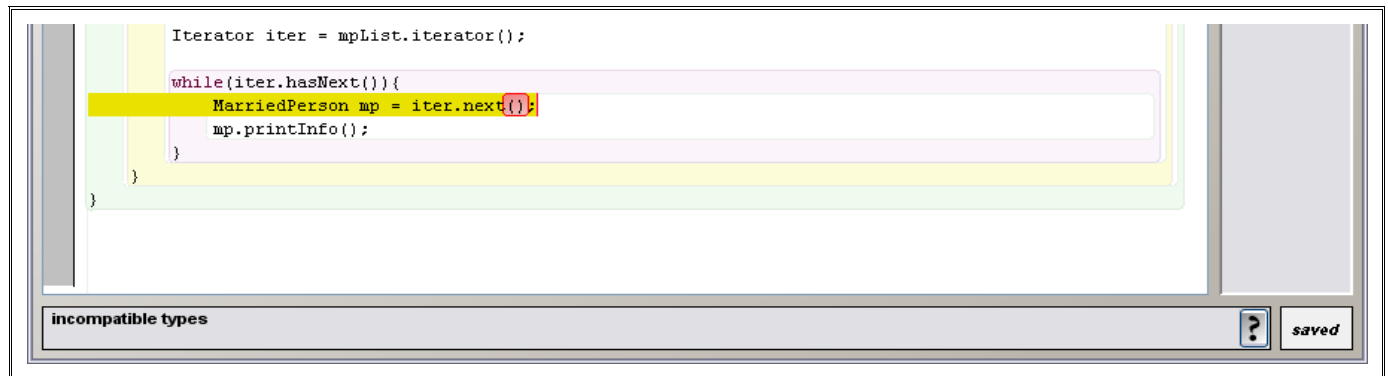
```
ArrayList<MarriedPerson> mpList=new ArrayList<MarriedPerson>();  
mpList.add(mp1);  
mpList.add(mp2);  
mpList.add(mp3);  
mpList.get(0).printInfo();  
mpList.get(1).printInfo();  
mpList.get(2).printInfo();
```

Τα αποτελέσματα που επιβεβαιώνουν τη δημιουργία της λίστας είναι τα εξής:



Το βασικό πλεονέκτημα μιας λίστας σε σχέση με τη δημιουργία πίνακα είναι πως μια λίστα είναι δυναμική και το μήκος της μπορεί να μεταβληθεί. Εν αντιθέσει με ένα πίνακα, του οποίου το μέγεθος πρέπει να γνωρίζουμε κατά τη στιγμή της δημιουργίας τους, γεγονός που μας δεσμεύει και δεν είναι ιδιαίτερα χρήσιμο και ως προς την αξιοποίηση της μνήμης και ως προς τη διαχείριση των αντικειμένων.

γ. Ακολουθώντας τις οδηγίες της εκφώνησης ώστε να διαπεράσουμε όλα τα στοιχεία της λίστας `mpList` που δημιουργήσαμε, προκύπτει το ακόλουθο σφάλμα:



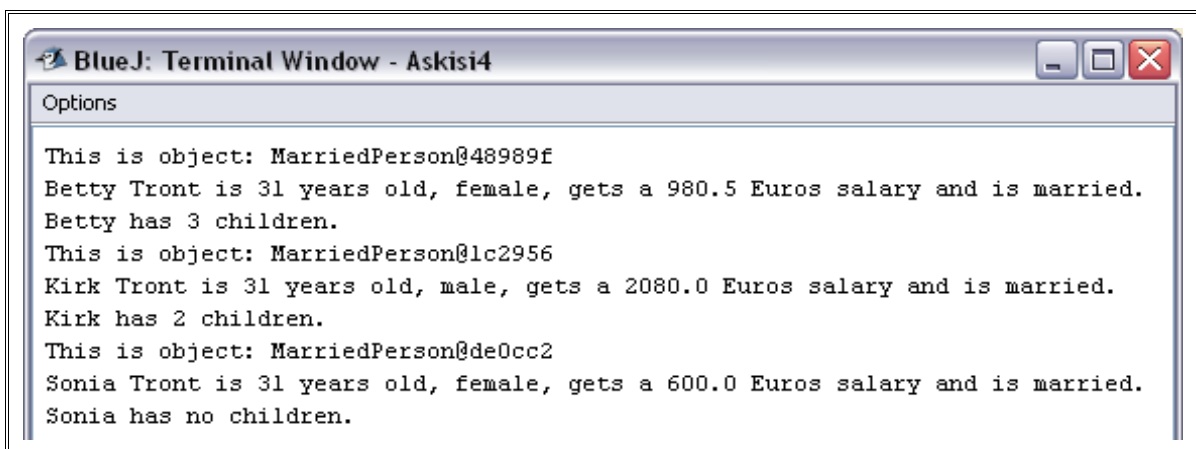
Δηλαδή, είναι ασύμβατος ο τύπος του `iterator` με τον τύπο της λίστας, κι αυτό συμβαίνει διότι οι απλοί `iterator` είναι τύπου `Object`, γι' αυτό και πρέπει να κάνουμε `cast` ώστε να γίνει σωστά, όπως προτείνεται και στην εκφώνηση.

δ. Δημιουργήσαμε τη λίστα έτσι εξ αρχής, συνεπώς το `cast` φαίνεται να είναι απαραίτητο ακόμα.

ε. Χρησιμοποιώντας την επαυξημένη μορφή της `for` για να προσπελάσουμε τα στοιχεία της `mpList`, θα προσθέταμε τον παρακάτω κώδικα:

```
for(MarriedPerson mp : mpList){
    System.out.println("This is object: " + mp);
    mp.printInfo();
}
```

Τα αποτελέσματα θα ήταν τα ακόλουθα:

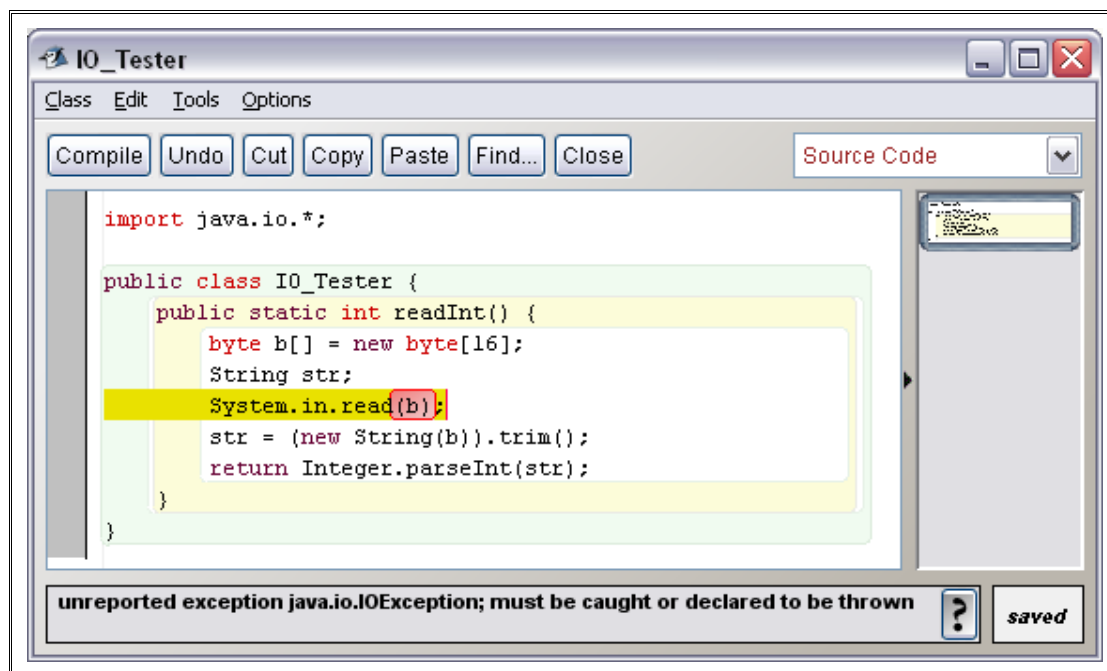


Τώρα, ο `iterator` καθίσταται αναγκαίος όταν θέλουμε να προσπελάσουμε τα στοιχεία από το τέλος προς την αρχή, όταν θέλουμε να κάνουμε σύγκριση ανάμεσα σε δύο στοιχεία, όταν θέλουμε να επεξεργαστούμε παραπάνω από μία δομές ή λίστες, όταν θέλουμε να διαγράψουμε κάποιο στοιχείο από

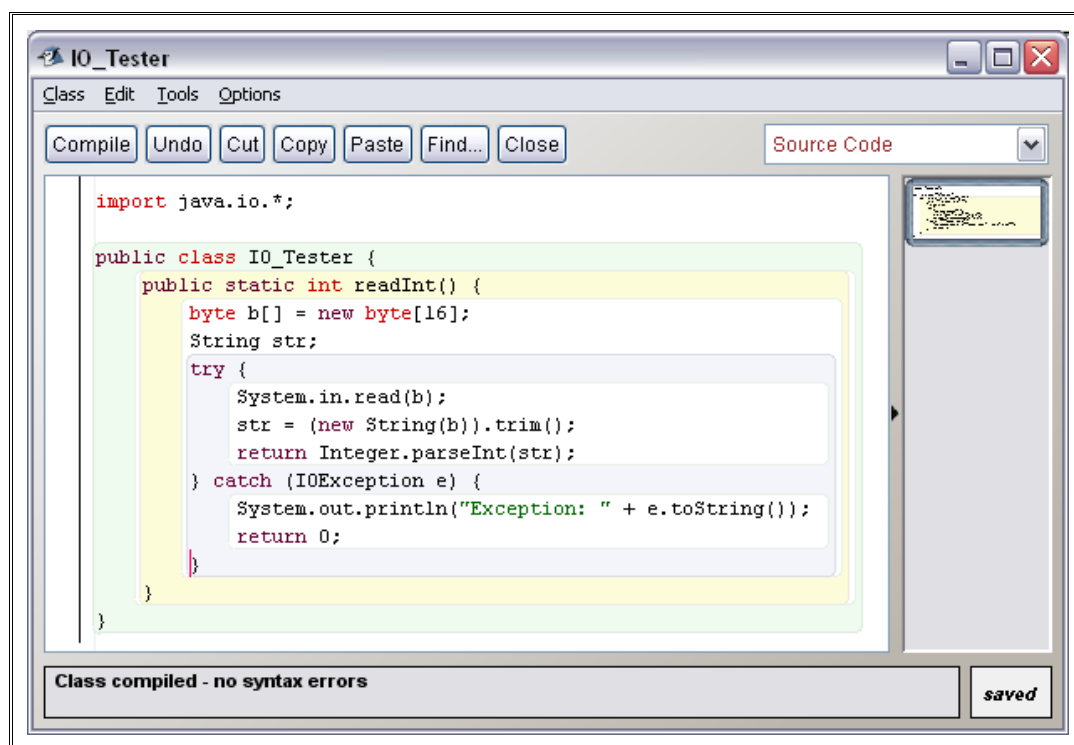
τη λίστα ή να επεξεργαστούμε τα δεδομένα ενός, εργασίες που δε γίνονται με χρήση της επαυξημένης μορφής της `for`.

5^η Εργαστηριακή Άσκηση

1) β. Μετά το σώσιμο και κατά τη μετάφραση, προέκυψε σφάλμα και ο μεταφραστής έβγαλε το ακόλουθο μήνυμα λάθους:

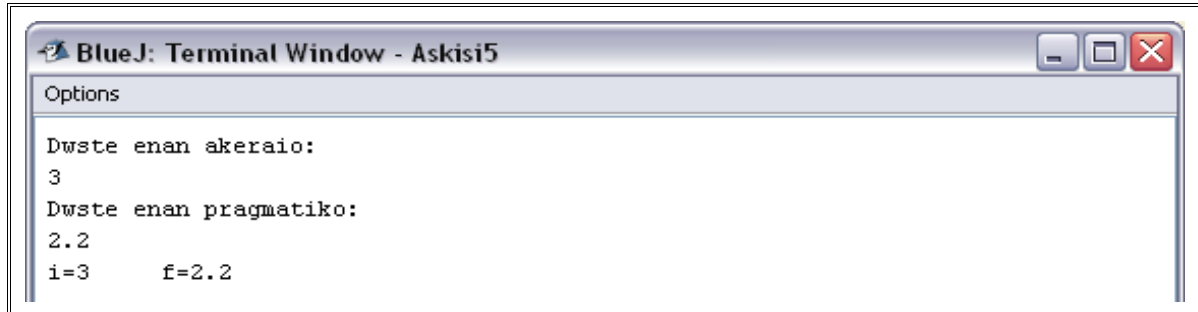


γ. Αντικαθιστώντας τον κώδικα όπως μας υποδεικνύεται στην εκφώνηση δεν εμφανίζεται λάθος:



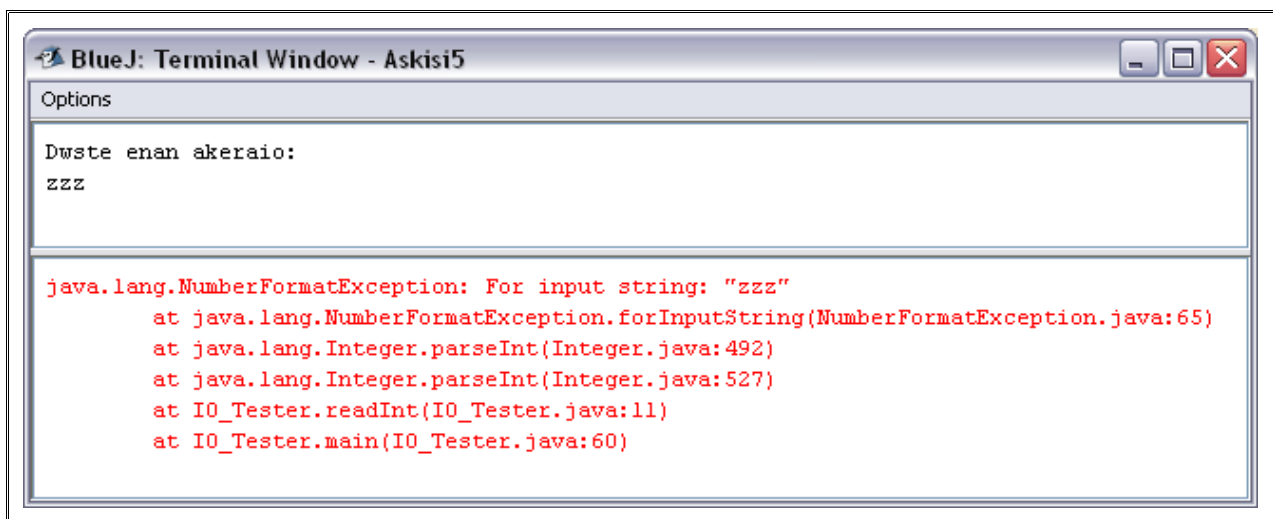
δ. Έπειτα από την προσθήκη των δοθέντων κωδίκων το πρόγραμμα μεταφράστηκε κανονικά και δεν εμφάνισε κάποιο σφάλμα ή μήνυμα λάθους.

2) Αφού κατασκευάσαμε τη `main` και μεταφράσαμε τον κώδικα, τα αποτελέσματα για την πρώτη φορά εκτέλεσής του είναι τα ακόλουθα:



```
Options
Dwste enan akeraio:
3
Dwste enan pragmatiko:
2.2
i=3    f=2.2
```

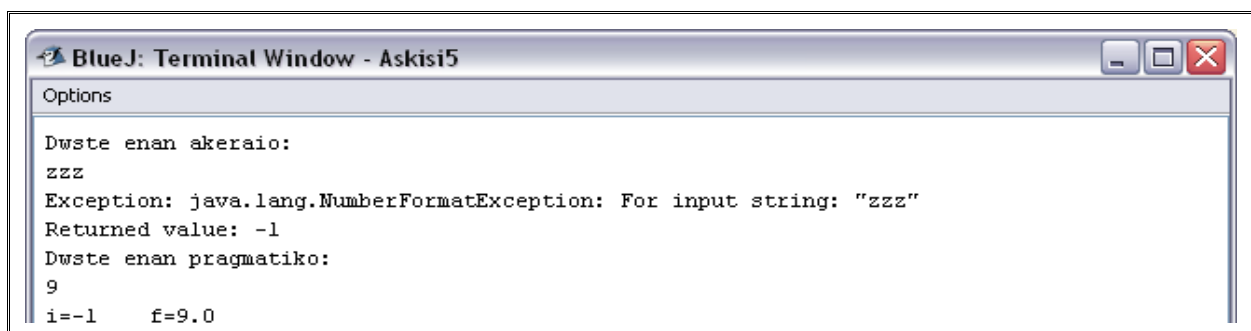
Για τη δεύτερη φορά εκτέλεσης του προγράμματος τα αποτελέσματα είναι τα εξής:



```
Options
Dwste enan akeraio:
zzz

java.lang.NumberFormatException: For input string: "zzz"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:492)
    at java.lang.Integer.parseInt(Integer.java:527)
    at IO_Tester.readInt(IO_Tester.java:11)
    at IO_Tester.main(IO_Tester.java:60)
```

3) Αντικαθιστώντας τον κώδικα της `readInt()` και μεταφράζοντας και τρέχοντας ξανά το πρόγραμμα, πήραμε τα παρακάτω αποτελέσματα:



```
Options
Dwste enan akeraio:
zzz
Exception: java.lang.NumberFormatException: For input string: "zzz"
Returned value: -1
Dwste enan pragmatiko:
9
i=-1    f=9.0
```

Παρατηρούμε τώρα ότι, σε αντίθεση με την προηγούμενη εκτέλεση, η έγερση της εξαίρεσης `NumberFormatException` δεν διακόπτει την εκτέλεση του προγράμματος, αλλά εμφανίζει ένα μήνυμα διαχείρισής της.

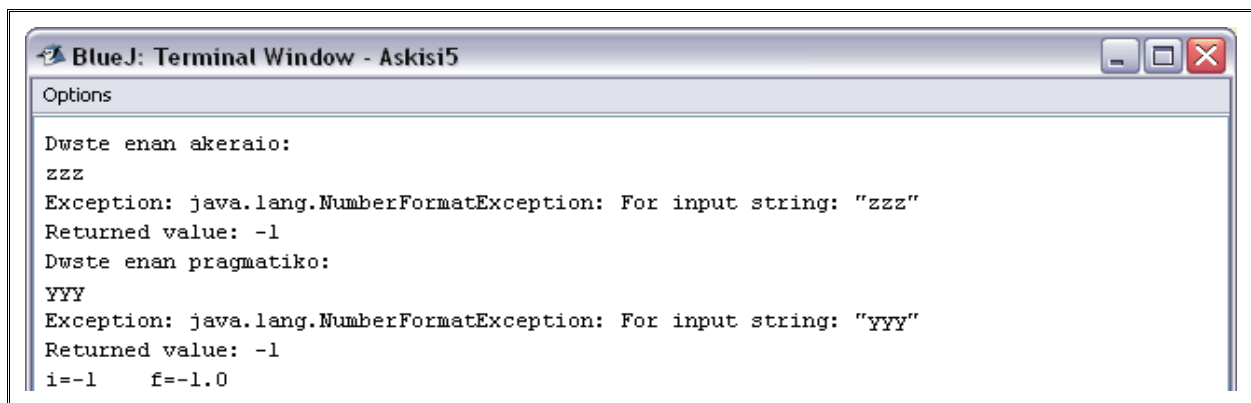
Για να διαχειριστούμε παρόμοια προβλήματα έγερσης εξαιρέσεων και στη μέθοδο `readFloat()`, μπορούμε να διαμορφώσουμε τον κώδικά της ως εξής:

```

public static float readFloat() {
    byte b[] = new byte[16];
    String str;
    try {
        System.in.read(b);
        str = (new String(b)).trim();
        return Float.parseFloat(str);
    } catch (IOException e) {
        System.out.println("Exception: " + e.toString());
        return 0;
    } catch (NumberFormatException e) {
        System.out.println("Exception: " + e.toString() +
            "\nReturned value: -1");
        return -1;
    }
}

```

Ξαναμεταφράζοντας το πρόγραμμα και εκτελώντας το, έχουμε τα εξής αποτελέσματα:



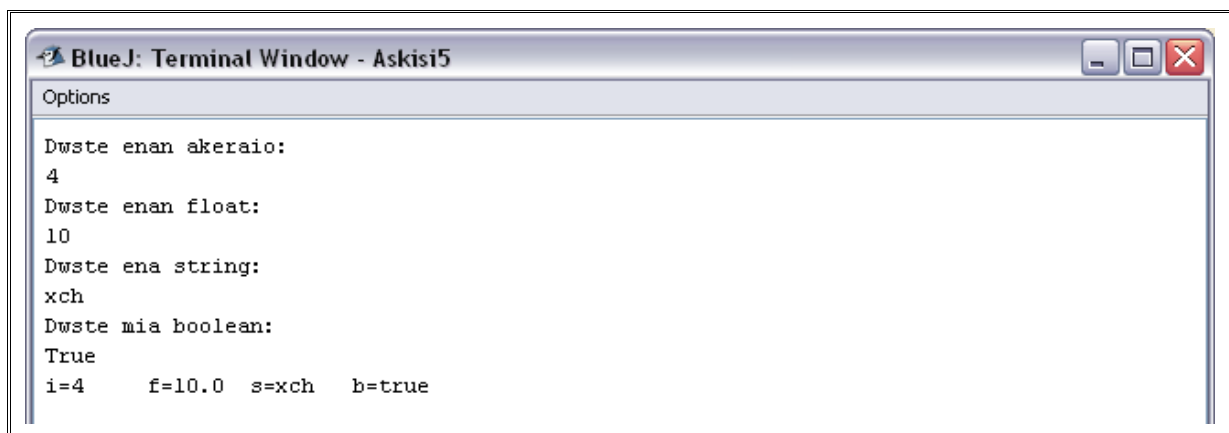
```

BlueJ: Terminal Window - Askisi5
Options
Dwste enan akeraio:
zzz
Exception: java.lang.NumberFormatException: For input string: "zzz"
Returned value: -1
Dwste enan pragmatiko:
YYY
Exception: java.lang.NumberFormatException: For input string: "yyy"
Returned value: -1
i=-1    f=-1.0

```

Παρατηρούμε ότι γίνεται αντίστοιχη διαχείριση της εξαίρεσης και σε περίπτωση σφάλματος στη μέθοδο `readFloat()`.

4) Στο ερώτημα αυτό τρέχουμε το πρόγραμμα τρεις φορές με τα δεδομένα που μας αναφέρονται. Έτσι, την πρώτη φορά και για τα δεδομένα «4», «10», «xch» και «True» έχουμε:



```

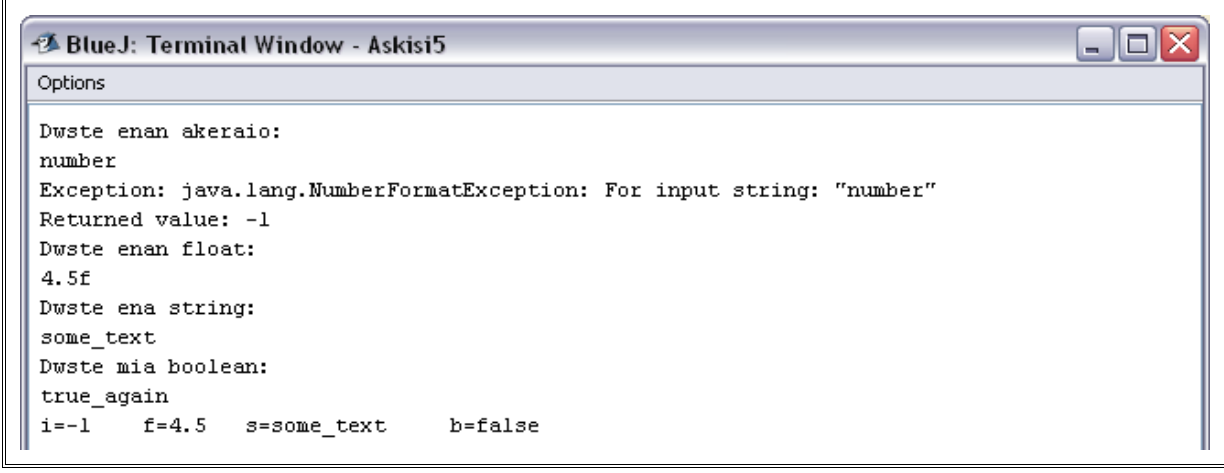
BlueJ: Terminal Window - Askisi5
Options
Dwste enan akeraio:
4
Dwste enan float:
10
Dwste ena string:
xch
Dwste mia boolean:
True
i=4    f=10.0  s=xch  b=true

```

Στην περίπτωση αυτή παρατηρούμε ότι το πρόγραμμα εκτελείται κανονικά, καθώς τα ορίσματα που δίνονται είναι σωστά και σύμφωνα με τους αποδεκτούς τύπους κάθε μεθόδου που καλείται.

Συνεπώς, τα αποτελέσματα εμφανίζονται κανονικά στο τέλος, χωρίς κάποια διαφοροποίηση.

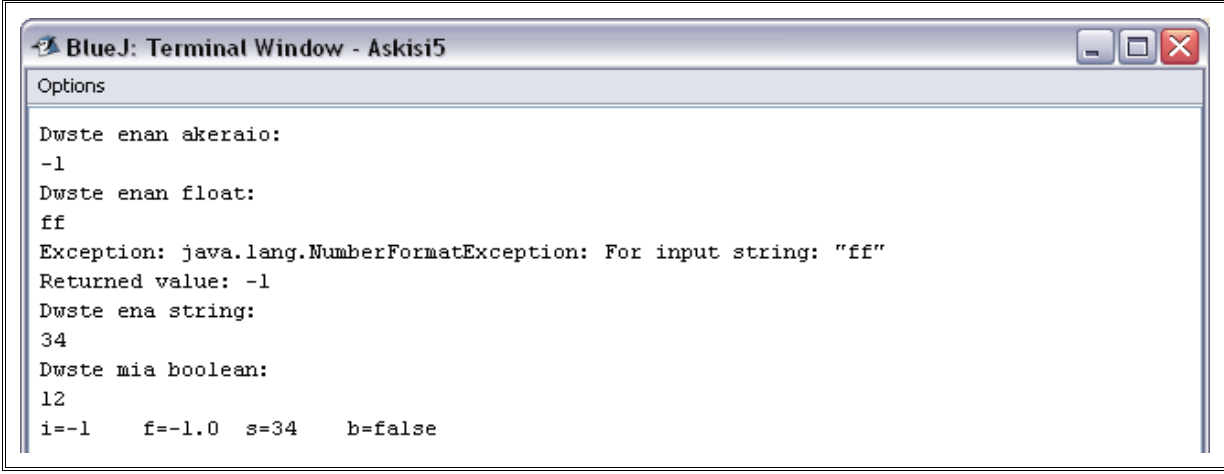
Εκτελώντας το πρόγραμμα για δεύτερη φορά, τώρα, και με τα ορίσματα «number», «4.5f», «some_text» και «true_again» έχουμε τα παρακάτω αποτελέσματα:



```
BlueJ: Terminal Window - Askisi5
Options
Dwste enan akeraio:
number
Exception: java.lang.NumberFormatException: For input string: "number"
Returned value: -1
Dwste enan float:
4.5f
Dwste ena string:
some_text
Dwste mia boolean:
true_again
i=-1    f=4.5    s=some_text    b=false
```

Στην περίπτωση αυτή παρατηρούμε ότι εγείρεται μια εξαίρεση, συγκεκριμένα η `NumberFormatException`, καθώς αντί για ακέραιο νούμερο δίνεται μια σειρά χαρακτήρων (γραμμάτων), οπότε προκύπτει σφάλμα και επιστρέφεται η τιμή «-1». Επίσης, αντί για `boolean` τιμή στο τέλος δίνεται πάλι μια σειρά χαρακτήρων η οποία εκλαμβάνεται ως `false`, γι' αυτό και εκτυπώνεται έτσι παρακάτω.

Φτάνοντας στην τρίτη εκτέλεση του προγράμματος, και δίνοντας ως ορίσματα τα «-1», «ff», «34» και «12», παίρνουμε τα εξής αποτελέσματα:

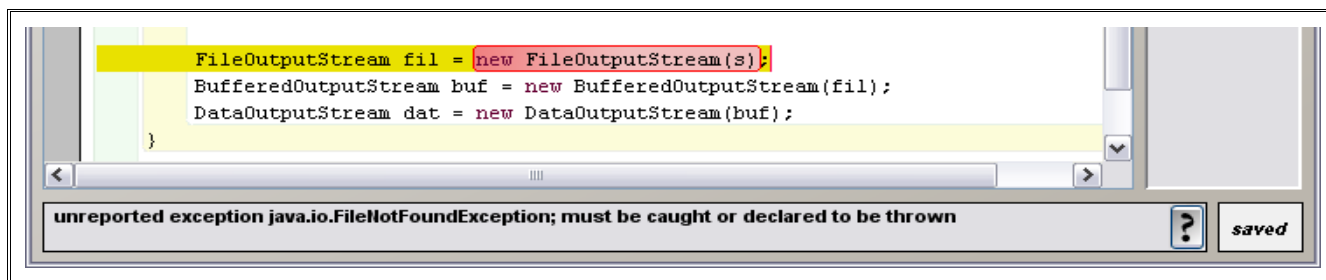


```
BlueJ: Terminal Window - Askisi5
Options
Dwste enan akeraio:
-1
Dwste enan float:
ff
Exception: java.lang.NumberFormatException: For input string: "ff"
Returned value: -1
Dwste ena string:
34
Dwste mia boolean:
12
i=-1    f=-1.0    s=34    b=false
```

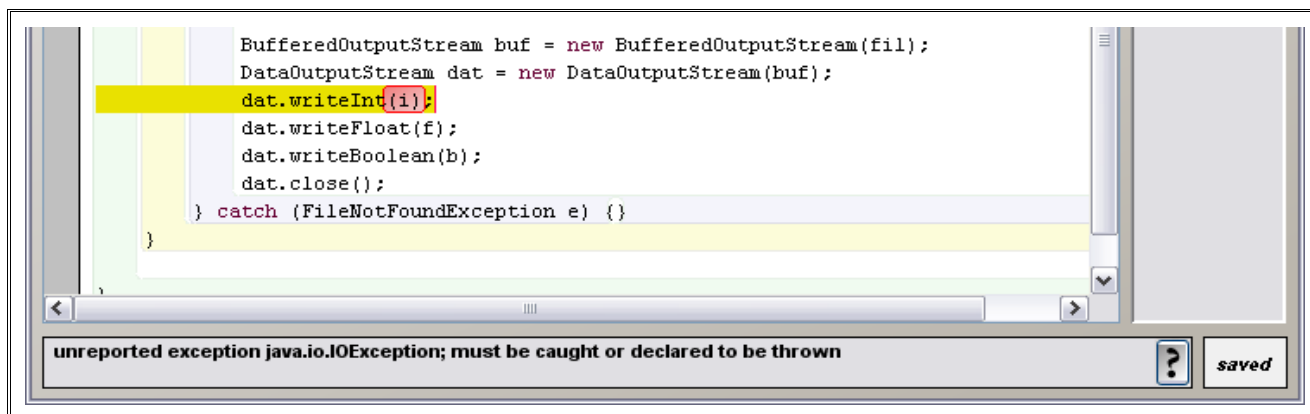
Στην τελευταία αυτή περίπτωση παρατηρούμε ότι πάλι έχουμε λανθασμένα ορίσματα σε δύο σημεία. Πρώτα, αντί για `float` αριθμός δίνεται μια σειρά χαρακτήρων (γραμμάτων), οπότε εγείρεται σωστά η εξαίρεση και επιστρέφεται η τιμή «-1». Έπειτα, αντί για `boolean` τιμή δίνεται ένας αριθμός διάφορος του 0 και του 1, οπότε εκλαμβάνεται ως `false`, και έτσι εκτυπώνεται και στο τέλος.

5) α. Προσθέτοντας το δοθέν κομμάτι κώδικα και κατά τη νέα μετάφρασή του, προέκυψε το παρακάτω σφάλμα:

(εικόνα στην επόμενη σελίδα)

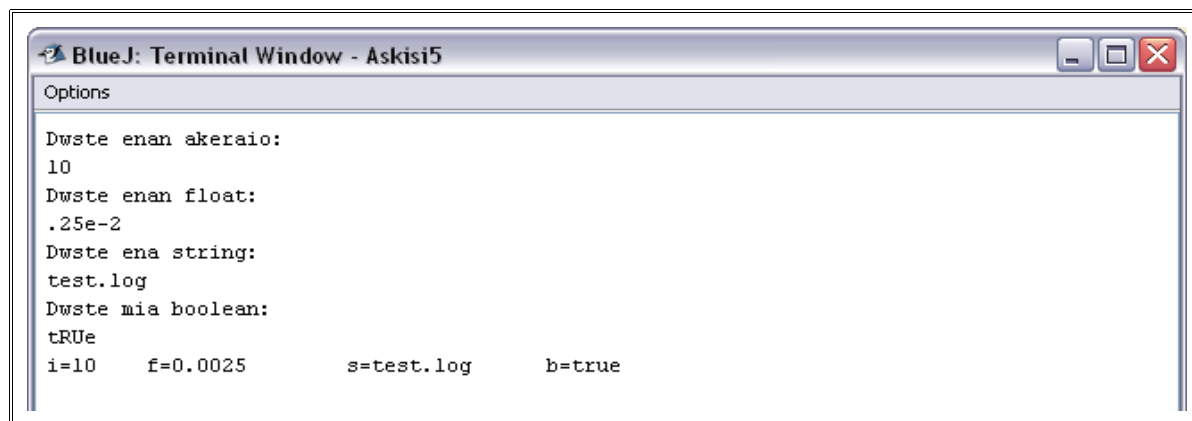


β. Αντικαθιστώντας τον παραπάνω κώδικα με τον δοθέντα σε αυτό το υποερώτημα, έχουμε πάλι σφάλμα, όπως φαίνεται και στην ακόλουθη εικόνα:

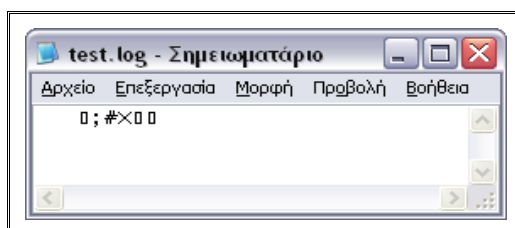


γ. Αντικαθιστώντας πάλι τον αντίστοιχο κώδικα, και επαναλαμβάνοντας τη μετάφραση του, παρατηρούμε όντως ότι δεν προκύπτει πλέον σφάλμα, κι ότι η διαδικασία της μετάφρασης έγινε σωστά.

δ. Προσθέτοντας τη γραμμή που μας δίνεται, μεταφράζοντας και τρέχοντας το πρόγραμμα για τα δοθέντα ορίσματα, έχουμε τα εξής αποτελέσματα:

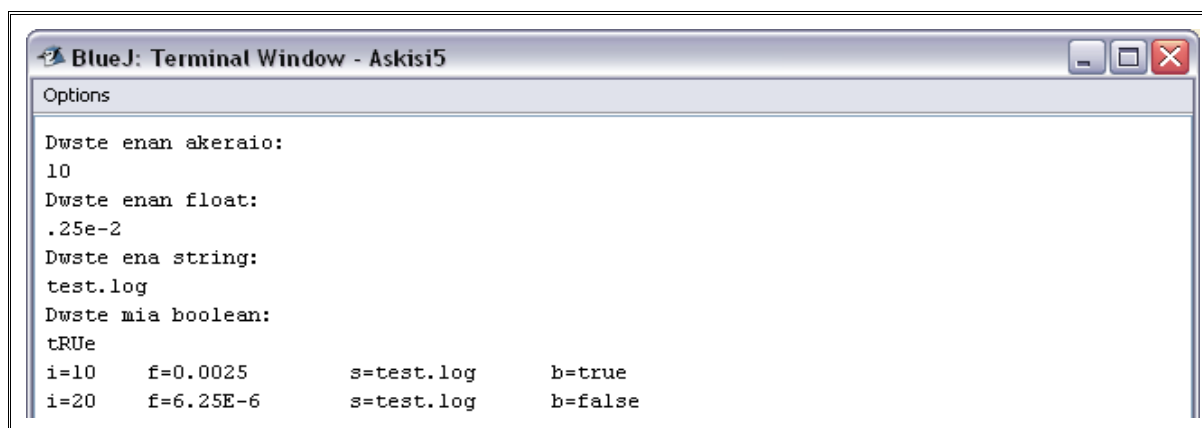


ε. Κοιτώντας στον κατάλογο του project για το αρχείο test.log και ανοίγοντάς του, βλέπουμε τα εξής περιεχόμενα σε αυτό:



Δυστυχώς, τα περιεχόμενά του δε βγάζουν κάποιο νόημα και αυτό πιθανόν να οφείλεται στο γεγονός ότι χρησιμοποιείται το `DataOutputStream` (δηλαδή χρήση stream δεδομένων), το οποίο θα γράψει τα δεδομένα ως μεμονωμένα bytes και όχι όπως όντως είναι. Έτσι, αυτό που καταλήγει να υπάρχει στο αρχείο δεν είναι άμεσα κατανοητό.

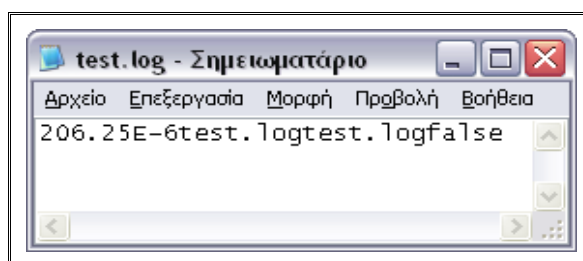
6) β. Τρέχοντας πάλι το πρόγραμμα μετά την αλλαγή του πρώτου υποερωτήματος, και βάζοντας τα ίδια δεδομένα, παίρνουμε τα ακόλουθα αποτελέσματα:



```
Options
Dwste enan akeraio:
10
Dwste enan float:
.25e-2
Dwste ena string:
test.log
Dwste mia boolean:
true
i=10    f=0.0025    s=test.log    b=true
i=20    f=6.25E-6  s=test.log    b=false
```

Ουσιαστικά, ο κώδικας που προστέθηκε αλλάζει τις τιμές των μεταβλητών `i`, `f` και `b` με τη βοήθεια του διαβάσματος από το αρχείο `test.log` που δημιουργείται, και εκτυπώνει τις νέες τους τιμές στο τέλος.

δ. Τρέχοντας το πρόγραμμα μετά την αλλαγή του προηγούμενου υποερωτήματος και για τα ίδια δεδομένα με πριν, βλέπουμε ότι τα περιεχόμενα του αρχείου `test.log` είναι τα παρακάτω:

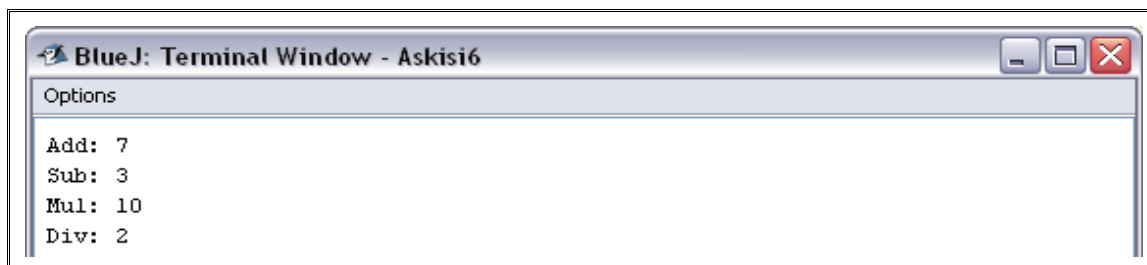


Ο παραπάνω κώδικας δουλεύει σωστά, καθώς πλέον χρησιμοποιείται ένα stream χαρακτήρων `FileWriter` και ένας αντίστοιχος buffer ώστε να γραφτούν εξίσου σωστά τα δεδομένα (ως χαρακτήρες) στο αρχείο `test.log`. Τα δεδομένα γράφονται κάθε φορά με την κλήση της μεθόδου `"bw.write(,,);"` και με τα ανάλογα ορίσματα, π.χ. `«Integer.toString(i)»` για να δεχτεί ως χαρακτήρες τον ακέραιο αριθμό που είναι αποθηκευμένος στη μεταβλητή `i`, `«0»` για να υποδεικνύεται το σημείο από όπου θα αρχίσουν να γράφονται τα δεδομένα, και `«Integer.toString(i).length()»` για να καθορίσει το μήκος του αριθμού `i` ως σειρά χαρακτήρων.

Ένα ενδιαφέρον σημείο είναι στην τρίτη κλήση της μεθόδου, για το `String s`, όπου χρησιμοποιείται η μέθοδος `concat()`, που στην προκειμένη περίπτωση ενώνει το αρχικό string `s` με τον εαυτό του, γι' αυτό και εμφανίζεται στο αρχείο δύο φορές συνεχόμενα.

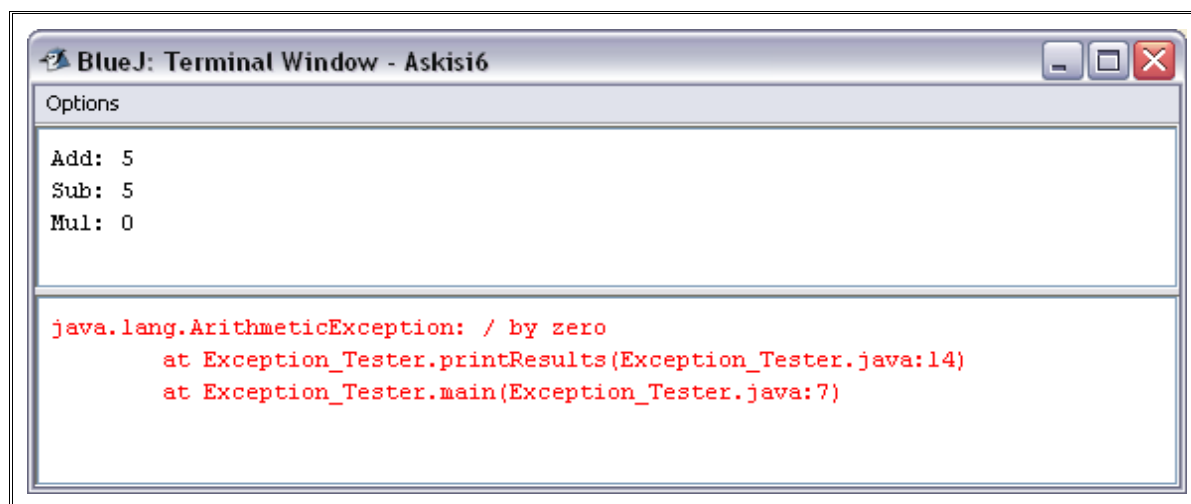
6^η Εργαστηριακή Άσκηση

1) β. Έπειτα από τη μετάφραση του προγράμματος, εκτελώντας το παίρνουμε τα παρακάτω:



```
BlueJ: Terminal Window - Askisi6
Options
Add: 7
Sub: 3
Mul: 10
Div: 2
```

γ. Αρχικοποιώντας τη μεταβλητή `x2` με την τιμή 0, ξαναμεταφράζοντας και τρέχοντας το πρόγραμμα πάλι, παίρνουμε τα παρακάτω αποτελέσματα:

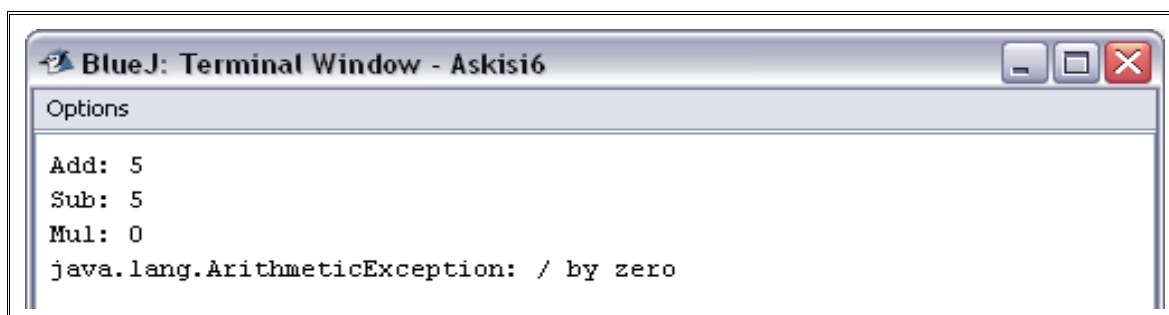


```
BlueJ: Terminal Window - Askisi6
Options
Add: 5
Sub: 5
Mul: 0

java.lang.ArithmeticException: / by zero
    at Exception_Tester.printResults(Exception_Tester.java:14)
    at Exception_Tester.main(Exception_Tester.java:7)
```

Παρατηρούμε ότι μόλις φτάνει στην πράξη της διαίρεσης εγείρεται μια εξαίρεση, συγκεκριμένα η `ArithmeticException`, καθώς δεν είναι δυνατή η διαίρεση με το μηδέν (μετά την αρχικοποίηση της `x2` που κάναμε παραπάνω).

2) Κάνοντας την αλλαγή που μας υποδεικνύεται στην εκφώνηση και τρέχοντας ξανά το πρόγραμμα, παίρνουμε τα εξής αποτελέσματα:

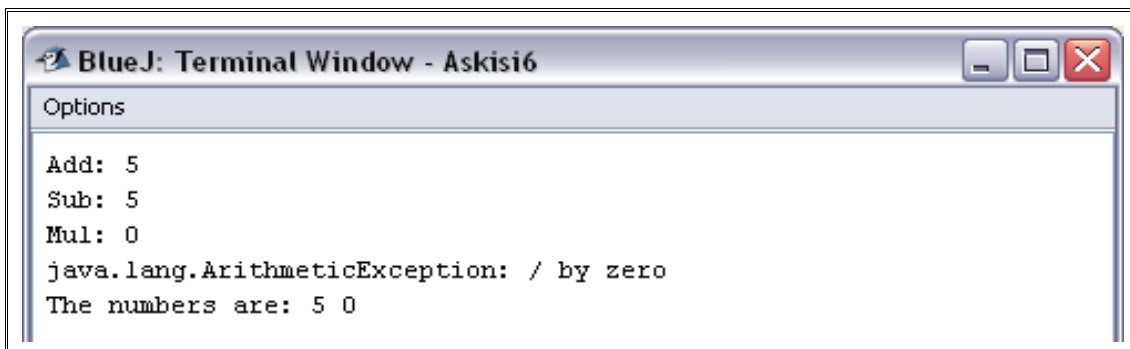


```
BlueJ: Terminal Window - Askisi6
Options
Add: 5
Sub: 5
Mul: 0
java.lang.ArithmeticException: / by zero
```

Παρατηρούμε τώρα ότι η διαχείριση της εξαίρεσης άλλαξε από πριν, καθώς πλέον εκτυπώνεται η

πρώτη γραμμή του μηνύματος που μας εμφανίστηκε προηγουμένως κατά την έγερσή της.

3) Αλλάζοντας τα κομμάτια κώδικα που μας δόθηκαν σύμφωνα με την εκφώνηση, και εκτελώντας εκ νέου το πρόγραμμα, παίρνουμε τα εξής αποτελέσματα:

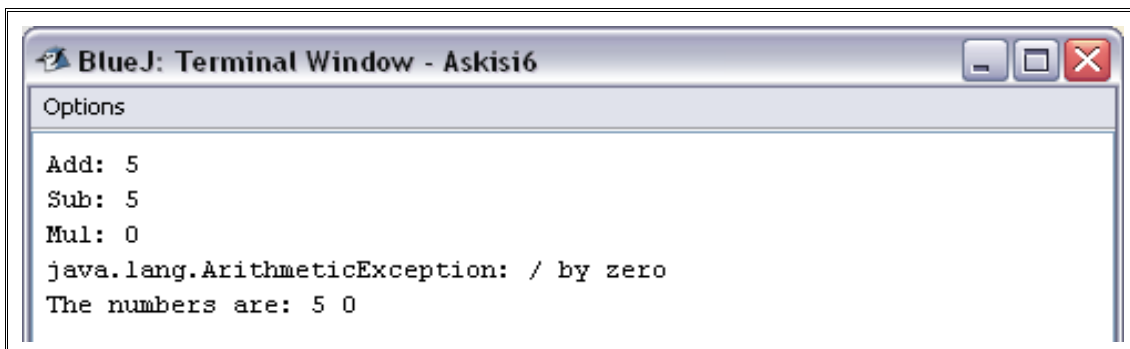


```
BlueJ: Terminal Window - Askisi6
Options
Add: 5
Sub: 5
Mul: 0
java.lang.ArithmeticException: / by zero
The numbers are: 5 0
```

Ουσιαστικά βλέπουμε πως έμειναν ίδια τα αποτελέσματα (εκτός βέβαια της γραμμής που εκτυπώνεται στο τέλος και προστέθηκε τώρα), αλλά άλλαξε ο τρόπος χειρισμού της εξαίρεσης. Αντί να τη διαχειρίζεται και να τη συλλαμβάνει κατά την κλήση της μεθόδου “`System.out.println("Div: " + (a/b)) ;`”, το κάνει κατά την κλήση της `printResults()` μέσα στο σώμα της `main`.

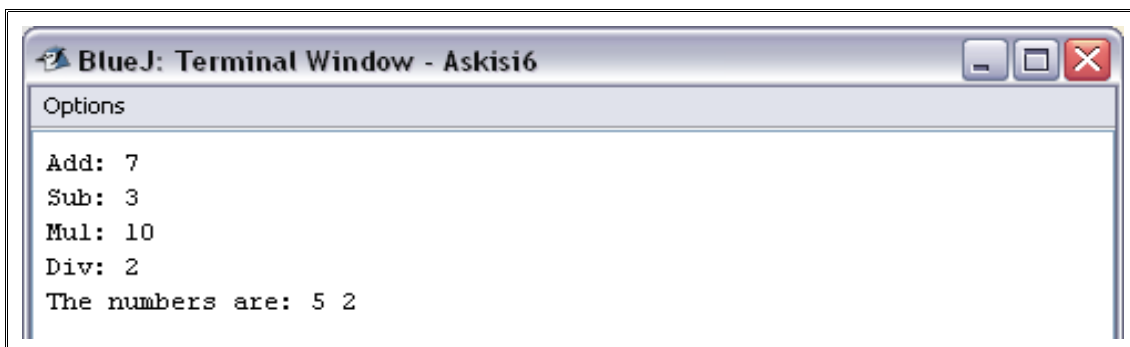
4) Δεν έχει συμπεριλάβει στον κώδικά της `main` του `try` και `catch` για τη διαχείριση της εξαίρεσης, παρόλο που επισημαίνει την πιθανότητα έγερσής της.

5) α. Έπειτα από την αντικατάσταση του κώδικα και την εκτέλεση του προγράμματος, παίρνουμε τα εξής αποτελέσματα:



```
BlueJ: Terminal Window - Askisi6
Options
Add: 5
Sub: 5
Mul: 0
java.lang.ArithmeticException: / by zero
The numbers are: 5 0
```

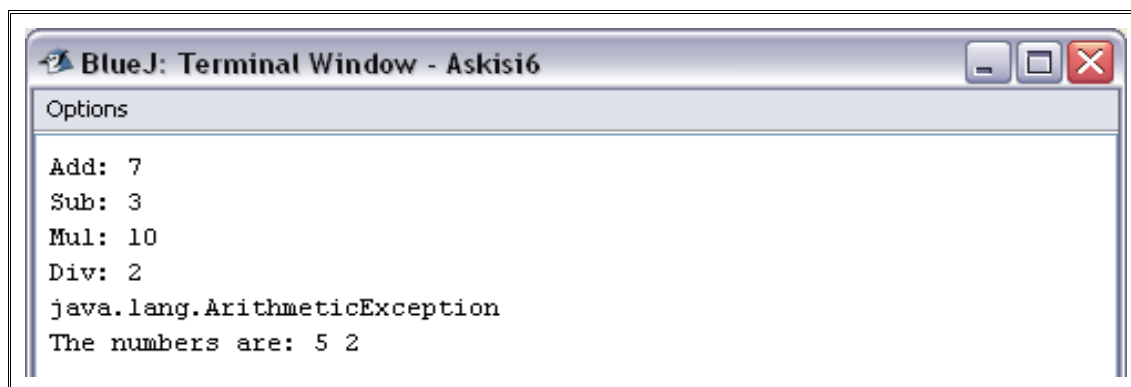
β. Αρχικοποιώντας πάλι την `x2` με την τιμή 0, μεταφράζοντας και εκτελώντας πάλι το πρόγραμμα, παίρνουμε τα ακόλουθα αποτελέσματα:



```
BlueJ: Terminal Window - Askisi6
Options
Add: 7
Sub: 3
Mul: 10
Div: 2
The numbers are: 5 2
```


γ. Σχετικά με τη λειτουργία της πρότασης `finally`, και λαμβάνοντας υπ' όψιν τα αποτελέσματα των δύο προηγούμενων υποερωτημάτων, μπορούμε να πούμε ότι εκτελείται πάντα ο κώδικας που βρίσκεται μέσα στο σώμα της, ανεξάρτητα από τον τρόπο με τον οποίο ο έλεγχος βγήκε από το συνολικό `try` block (π.χ. με έγερση εξαίρεσης, με `break`, κτλ.). Έτσι, πάντα εκτυπώνεται στο τέλος η πρόταση “The numbers are: _ _”.

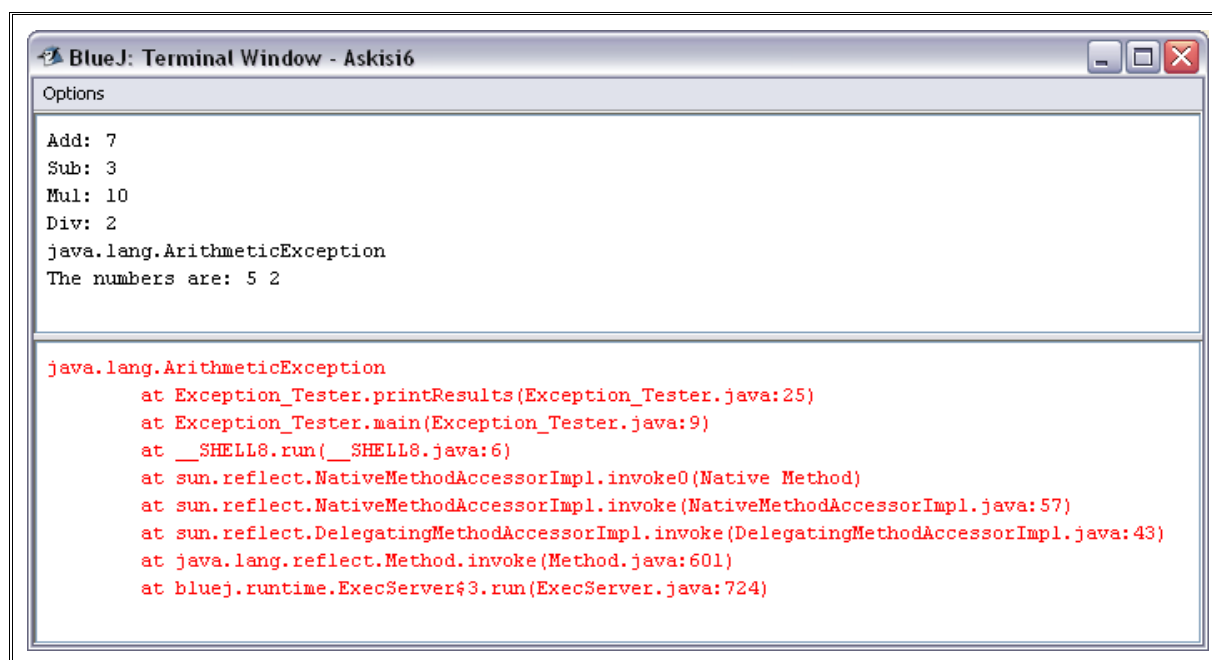
6) α. Μετά την προσθήκη του κώδικα και την εκτέλεση του προγράμματος, έχουμε τα εξής:



```
BlueJ: Terminal Window - Askisi6
Options
Add: 7
Sub: 3
Mul: 10
Div: 2
java.lang.ArithmeticException
The numbers are: 5 2
```

Στην ουσία ο κώδικας που προστέθηκε προκαλεί μια έγερση εξαίρεσης, ανεξάρτητα από την ομαλή εκτέλεση ή όχι του προγράμματος. Στη συγκεκριμένη περίπτωση, εγείρει μια `ArithmeticException`.

β. Μετά την προσθήκη και της νέας γραμμής κώδικα και την επανεκτέλεση του προγράμματος, έχουμε τα εξής αποτελέσματα:



```
BlueJ: Terminal Window - Askisi6
Options
Add: 7
Sub: 3
Mul: 10
Div: 2
java.lang.ArithmeticException
The numbers are: 5 2

java.lang.ArithmeticException
    at Exception_Tester.printResults(Exception_Tester.java:25)
    at Exception_Tester.main(Exception_Tester.java:9)
    at __SHELL8.run(__SHELL8.java:6)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:601)
    at bluej.runtime.ExecServer$3.run(ExecServer.java:724)
```

Παρατηρούμε ότι συνέβη πάλι έγερση της ίδιας εξαίρεσης, μόνο που αυτή τη φορά εκτυπώθηκε στην περιοχή μηνυμάτων του BlueJ (ή πιο ακριβέστερα, στο `System.err`) η στοίβα με όλα τα στοιχεία της εξαίρεσης. Ουσιαστικά μας λέει τι συνέβη, ποια εξαίρεση εγέρθηκε και πού μέσα στον κώδικά μας.

7) α. Κάνοντας τις αλλαγές κώδικα όπως μας υποδεικνύεται από την εκφώνηση, έχουμε:

```
BlueJ: Terminal Window - Askisi6
Options
Add: 7
Sub: 3
Mul: 10
Div: 2
DivideByZeroException
The numbers are: 5 2

DivideByZeroException
    at Exception_Tester.printResults(Exception_Tester.java:25)
    at Exception_Tester.main(Exception_Tester.java:9)
    at __SHELL9.run(__SHELL9.java:6)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:601)
    at bluej.runtime.ExecServer$3.run(ExecServer.java:724)
```

Αυτό που άλλαξε στα αποτελέσματα είναι ότι πλέον δημιουργήσαμε μια νέα κλάση `DivideByZeroException` για διαχείριση `ArithmeticException`, αφού είναι και υποκλάση της. Στην ουσία δεν άλλαξε το είδος της εξαίρεσης, απλά πλέον είναι στην ευχέρεια του προγραμματιστή να καθορίσει ακριβέστερα πώς να διαχειρίζεται τις εγέρσεις αυτών των εξαιρέσεων.

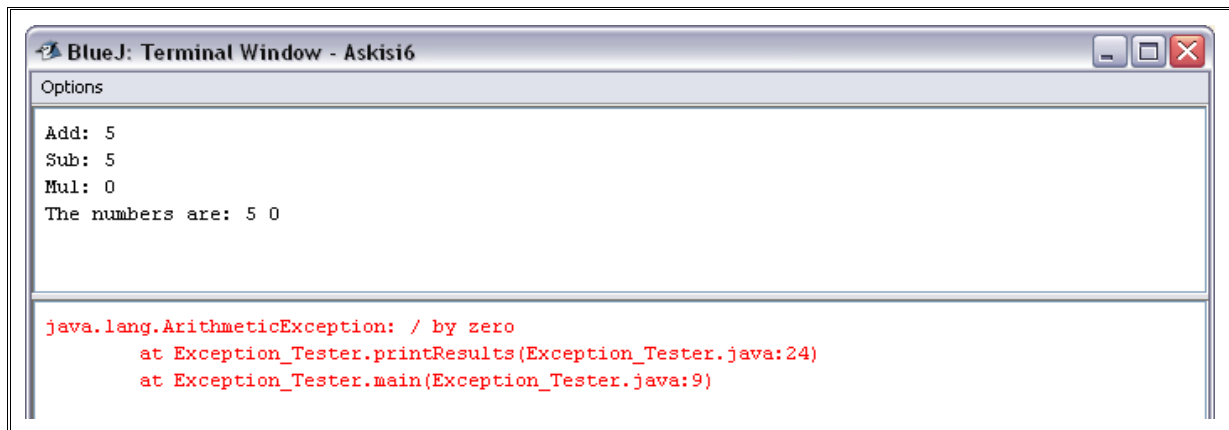
β. Ακολουθώντας τις υποδείξεις της εκφώνησης, κι αφού κάνουμε τις ανάλογες αλλαγές, έχουμε:

```
BlueJ: Terminal Window - Askisi6
Options
Add: 7
Sub: 3
Mul: 10
Div: 2
DivideByZeroException: The denominator cannot be zero.
The numbers are: 5 2

DivideByZeroException: The denominator cannot be zero.
    at Exception_Tester.printResults(Exception_Tester.java:25)
    at Exception_Tester.main(Exception_Tester.java:9)
    at __SHELL10.run(__SHELL10.java:6)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:601)
    at bluej.runtime.ExecServer$3.run(ExecServer.java:724)
```

γ. Αρχικοποιώντας τη μεταβλητή `x2` στην τιμή 0, έχουμε τα αποτελέσματα:

(εικόνα στην επόμενη σελίδα)

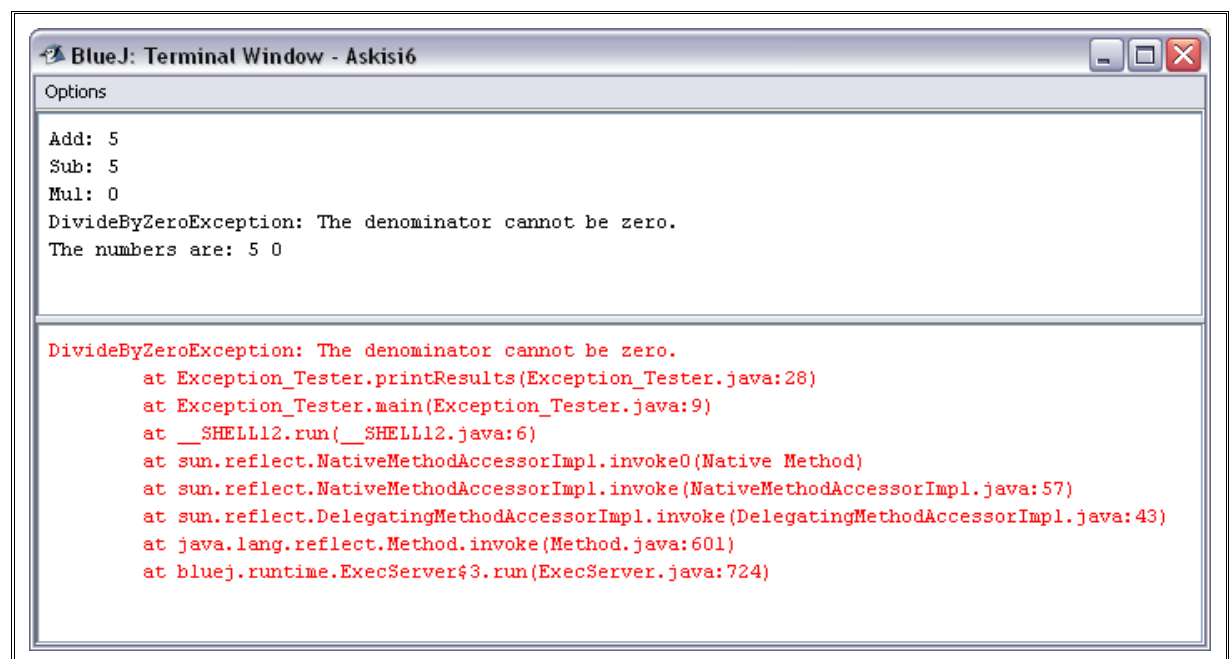


```
BlueJ: Terminal Window - Askisi6
Options
Add: 5
Sub: 5
Mul: 0
The numbers are: 5 0

java.lang.ArithmeticException: / by zero
    at Exception_Tester.printResults(Exception_Tester.java:24)
    at Exception_Tester.main(Exception_Tester.java:9)
```

Βλέπουμε ότι προκύπτει `ArithmeticException` και στις δύο περιπτώσεις, παρόλο που έχουμε και την `DivideByZeroException`. Στην ουσία αυτό συμβαίνει γιατί δεν έχει οριστεί ακόμα σωστά η δεύτερη ώστε να διαχειρίζεται περιπτώσεις διαίρεσης με το μηδέν.

δ. Αντικαθιστώντας τον κώδικα της `printResults()`, και εκτελώντας πάλι το πρόγραμμα, έχουμε τα εξής αποτελέσματα:



```
BlueJ: Terminal Window - Askisi6
Options
Add: 5
Sub: 5
Mul: 0
DivideByZeroException: The denominator cannot be zero.
The numbers are: 5 0

DivideByZeroException: The denominator cannot be zero.
    at Exception_Tester.printResults(Exception_Tester.java:28)
    at Exception_Tester.main(Exception_Tester.java:9)
    at __SHELL12.run(__SHELL12.java:6)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:601)
    at bluej.runtime.ExecServer$3.run(ExecServer.java:724)
```

Παρατηρούμε ότι διαχειρίζεται την έγερση της `ArithmeticException` καλώντας την `DivideByZeroException` που έχουμε ορίσει. Κι έτσι έχουμε ως αποτέλεσμα την εκτύπωση του μηνύματος που ορίζεται μέσα στην `DivideByZeroException` και τα αντίστοιχα στοιχεία στην περιοχή μηνυμάτων.

1^η Άσκηση 2^{ου} Σετ Ασκήσεων

Ακολουθώντας τις οδηγίες της εκφώνησης, συντάξαμε τον ακόλουθο κώδικα. Έτσι, για την κλάση Employee έχουμε:

```
public abstract class Employee{
    private String name;
    private String afm;
    private long EmpID;
    private static int tempID=0;

    public Employee(){
        tempID=tempID+1;
        this.EmpID=tempID;
    }

    public void setName(String n){
        this.name=n;
    }

    public String getName(){
        return this.name;
    }

    public void setAfm(String a){
        this.afm=a;
    }

    public String getAfm(){
        return this.afm;
    }

    public long getEmpID(){
        return this.EmpID;
    }

    abstract int payment();
}
```

Για την κλάση SalariedEmployee έχουμε:

```
public class SalariedEmployee extends Employee{
    private int salary=0;

    void setSalary(int s){
        this.salary=s;
    }

    public int payment(){
        return this.salary;
    }
}
```

Για την κλάση HourlyEmployee έχουμε:

```
public class HourlyEmployee extends Employee{
    private int hoursWorked=0;
    private int hourlyPayment=0;

    public void setHoursWorked(int hw){
        this.hoursWorked=hw;
    }

    public int getHoursWorked(){
        return this.hoursWorked;
    }

    public void setHourlyPayment(int hp){
        this.hourlyPayment=hp;
    }

    public int getHourlyPayment(){
        return this.hourlyPayment;
    }

    public int payment(){
        return this.hoursWorked*this.hourlyPayment;
    }
}
```

Τέλος, για την κλάση που περιέχει τη main έχουμε:

```
public class Main{
    public static void main(String args[]){
        Employee emp_list[]=new Employee[2];

        emp_list[0]=new SalariedEmployee();
        emp_list[1]=new HourlyEmployee();

        emp_list[0].setName("Gewrgiou");
        emp_list[0].setAfm("777777");
        ((SalariedEmployee)emp_list[0]).setSalary(300000);

        emp_list[1].setName("Karamitros");
        emp_list[1].setAfm("888888");
        ((HourlyEmployee)emp_list[1]).setHoursWorked(3000);
        ((HourlyEmployee)emp_list[1]).setHourlyPayment(40);

        for(int i=0; i<2; i++){
            System.out.println("Employee ID: "+emp_list[i].getEmpID());
            System.out.println("Employee Name: "+emp_list[i].getName());
            System.out.println("Employee AFM: "+emp_list[i].getAfm());
            System.out.println("Employee Payment: "+emp_list[i].payment()
                               +"\n");
        }
    }
}
```

Έπειτα από τη μετάφραση και την εκτέλεση των παραπάνω, πήραμε τα ακόλουθα αποτελέσματα, τα οποία είναι όντως τα ζητούμενα:



```
BlueJ: Terminal Window - 1_Askisi_2oSet
Options
Employee ID: 1
Employee Name: Gewrgiou
Employee AFM: 777777
Employee Payment: 300000

Employee ID: 2
Employee Name: Karamitros
Employee AFM: 888888
Employee Payment: 120000
```

2^η Άσκηση 2^{ου} Σετ Ασκήσεων

Το πρόγραμμα JAVA που μας ζητείται ώστε να πραγματοποιείται η διαίρεση δύο ακεραίων, είναι το ακόλουθο:

```
import java.io.*;

public class Division{
    public static void main(String args[]){
        System.out.print("Please insert the first number: ");
        int first=readInt();
        System.out.print("Please insert the second number: ");
        int second=readInt();

        try{
            if((first<0) || (second<0)){
                System.out.println("The division couldn't be done.");
            }else{
                System.out.println("The result after the division is"
                                   +divide_x_y(first,second)+".");
            }
        }catch(ArithmeticException ae){
            System.out.println("ArithmeticException: cannot divide with
                               zero.");
        }
    }

    public static int readInt(){
        byte number[]=new byte[16];
        String temp_str;

        try{
            System.in.read(number);
            temp_str=(new String(number)).trim();
            return Integer.parseInt(temp_str);
        }catch(IOException ioe){
            System.out.println("IOException: "+ioe.toString()+".");
            return 0;
        }catch(NumberFormatException nfe){
            System.out.println("NumberFormatException: "+nfe.toString()
                               +".");
            return -1;
        }
    }

    public static double divide_x_y(int x,int y) throws ArithmeticException{
        double result= (double)x/(double)y;

        if(result==Double.POSITIVE_INFINITY ||
           result==Double.NEGATIVE_INFINITY){
            throw new ArithmeticException("ArithmeticException: cannot
                                           divide with zero.");
        }
    }
}
```

```

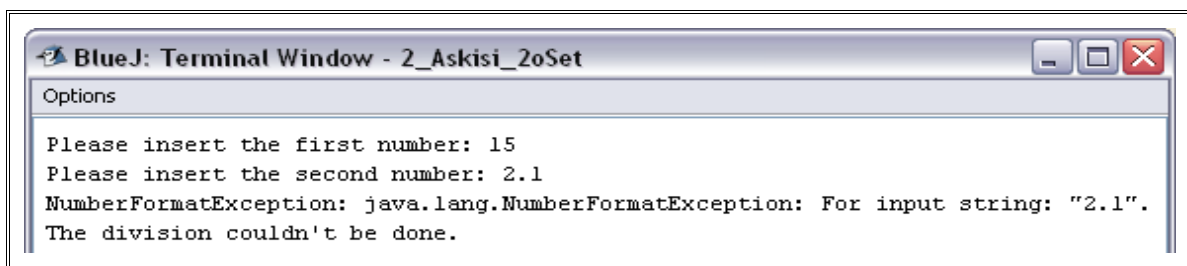
    }
    return result;
}
}

```

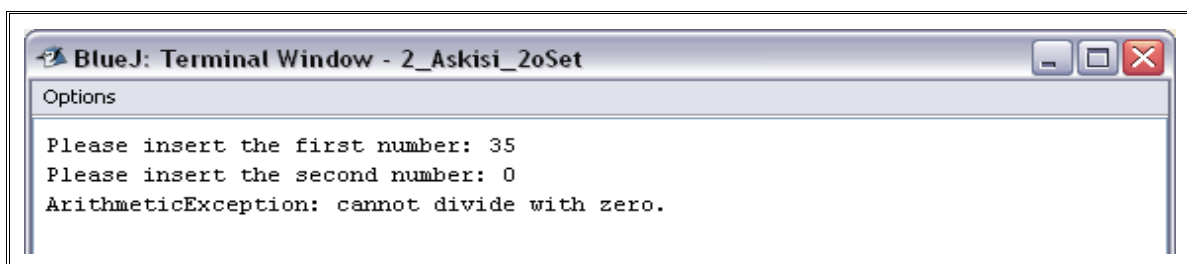
Η λειτουργία της `main` είναι προφανής, οπότε αξίζει να εξηγήσουμε λίγο καλύτερα την λειτουργία της `readInt()`. Αρχικά ορίζουμε έναν πίνακα `number[]`, τύπου `byte` και μεγέθους 16 κελιών, και μια μεταβλητή `String`. Στη συνέχεια, μέσα σε ένα `try-catch` block (ώστε να διαχειριζόμαστε τις εκάστοτε εξαιρέσεις) διαβάζουμε έναν αριθμό ως είσοδο από τον χρήστη με την `System.in.read()`. Την είσοδο αυτή την αποθηκεύουμε στο προσωρινό `String`, αφού εξαλείψουμε ενδεχόμενα κενά πριν ή μετά από την είσοδο, με τη βοήθεια της `trim()`. Τέλος, επιστρέφουμε μέσω της κλήσης της `Integer.parseInt()`, η οποία επιστρέφει με τη σειρά της το προηγούμενο `String` ως έναν προσημασμένο ακέραιο αριθμό. Όλη αυτή η διαδικασία γίνεται ώστε να καταστεί δυνατός ο έλεγχος της εισόδου για το αν είναι ακέραιος ή όχι.

Επίσης, ας ρίξουμε μια ματιά και στη λειτουργία της `divide_x_y()`. Δέχεται ως ορίσματα δύο ακραίους ώστε να πραγματοποιήσει τη διαίρεσή τους, της οποίας το αποτέλεσμα θα είναι `double`, λόγω του `typecasting`. Αν το αποτέλεσμα είναι το θετικό άπειρο ή το αρνητικό, τότε σημαίνει ότι ο διαιρέτης μας είναι ίσος με το μηδέν, οπότε εγείρεται η αντίστοιχη `ArithmeticException` με το κατάλληλο μήνυμα.

Για να επιβεβαιώσουμε την ορθή λειτουργία του προγράμματος, δίνουμε διάφορες εισόδους, κατάλληλες ώστε να εγείρονται οι αντίστοιχες εξαιρέσεις κάθε φορά. Οπότε, βάζοντας για είσοδο “15” και “2.1” (όπου ο δεύτερος αριθμός είναι `float`), έχουμε τα εξής αποτελέσματα:

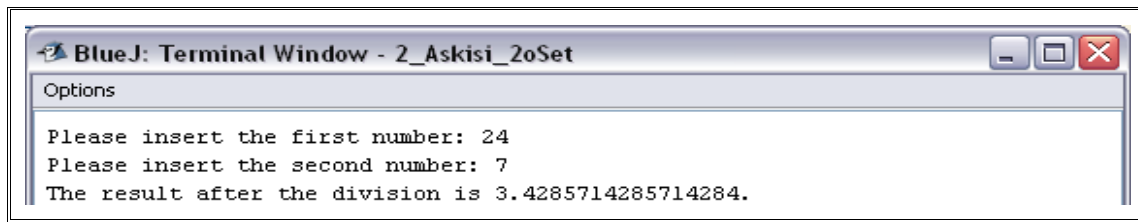


Παρατηρούμε ότι είναι όντως τα αναμενόμενα αποτελέσματα, με την έγερση της εξαίρεσης `NumberFormatException` και την εκτύπωση του κατάλληλου μηνύματος. Βάζοντας για είσοδο τώρα τους αριθμούς “35” και “0”, έχουμε τα ακόλουθα αποτελέσματα:



Παρατηρούμε ότι και πάλι λειτουργεί σωστά, μιας και εγείρεται η `ArithmeticException`, εξαιτίας του “0” που εισάγαμε, και εκτυπώνεται το ανάλογο μήνυμα για την αδυναμία της πραγματοποίησης της διαίρεσης. Βάζοντας, τέλος, για είσοδο τις τιμές “24” και “7”, παίρνουμε τα παρακάτω αποτελέσματα:

(εικόνα στην επόμενη σελίδα)



Παρατηρούμε ότι και εδώ τα αποτελέσματα είναι τα αναμενόμενα, καθώς και οι δύο είσοδοι είναι ακέραιοι, και ο διαιρέτης μη μηδενικός.

3^η Άσκηση 2^{ου} Σετ Ασκήσεων

Ακολουθώντας τις οδηγίες της εκφώνησης, συντάξαμε το παρακάτω πρόγραμμα ώστε να υλοποιεί όλες τις ζητούμενες λειτουργίες:

```
import java.io.*;

public class File_Exe{
    public static void main(String args[]){
        int temp;
        int counter=0;
        long src_length=0;
        long dest_length=0;

        try{
            File temp_file_1=new File("src.txt");
            boolean file_exists=temp_file_1.exists();
            FileInputStream file_input=new FileInputStream("src.txt");
        }catch(FileNotFoundException fnfe){
            System.out.println("The file \"src.txt\" wasn't found in this
                               directory.");
            System.exit(0);
        }

        try{
            File temp_file_1=new File("src.txt");
            File temp_file_2=new File("temp.txt");

            src_length=temp_file_1.length();
            dest_length=temp_file_2.length();

            FileReader file_reader=new FileReader(temp_file_1);
            FileWriter file_writer=new FileWriter(temp_file_2);

            System.out.println("Before copying \"src.txt\"
                               to \"dest.txt\":");
            System.out.println("\t-\"src.txt\" Length: "+src_length);
            System.out.println("\t-\"dest.txt\" Length:
                               "+dest_length+"\n");

            do{
                temp=file_reader.read();

                if(temp!=-1){
                    if((char)temp==' '){
                        temp=(int) '@';
                        counter++;
                    }

                    file_writer.write(temp);
                }
            }while(temp!=-1);
        }
```

```

        file_reader.close();
        file_writer.close();

    }catch(FileNotFoundException fnfe){
        System.out.println("FileNotFoundException:    "+fnfe.toString()
                            +".");
        System.exit(0);
    }catch(IOException ioe){
        System.out.println("IOException: "+ioe.toString()+".");
        System.exit(0);
    }

    try{
        File temp_file_1=new File("temp.txt");
        File temp_file_2=new File("dest.txt");
        String EOL=System.getProperty("line.separator");
        String line=null;

        FileReader file_reader=new FileReader(temp_file_1);
        FileWriter file_writer=new FileWriter(temp_file_2);
        BufferedReader file_buffer_read=new
            BufferedReader(file_reader);
        BufferedWriter file_buffer_write=new
            BufferedWriter(file_writer);

        while((line=file_buffer_read.readLine())!=null){
            file_buffer_write.write(line+EOL+EOL);
        }

        file_buffer_write.close();
        file_buffer_read.close();
        file_reader.close();
        file_writer.close();

        System.out.println("After copying \"src.txt\"
                           to \"dest.txt\":");
        System.out.println("\t-\"src.txt\" Length: "+src_length);
        System.out.println("\t-\"dest.txt\" Length:
                           "+temp_file_2.length()+"\n");
        System.out.println(counter+" spaces were replaced with the
                           character \"@\".");

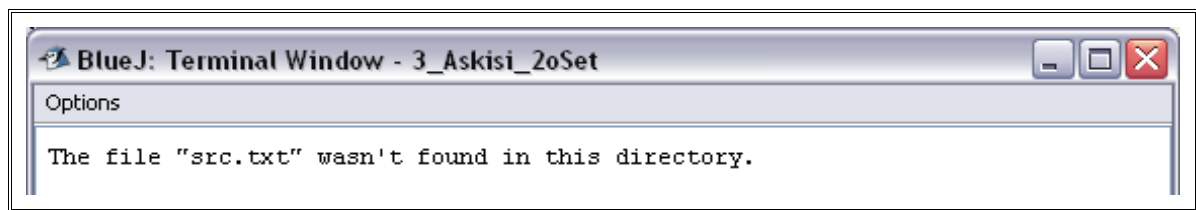
    }catch(FileNotFoundException fnfe){
        System.out.println("FileNotFoundException:    "+fnfe.toString()
                            +".");
        System.exit(0);
    }catch(IOException ioe){
        System.out.println("IOException: "+ioe.toString()+".");
        System.exit(0);
    }

}
}

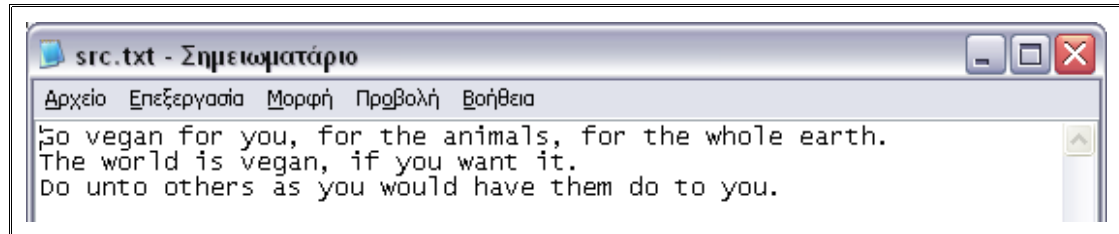
```

Εκτελώντας το πρόγραμμα χωρίς να υπάρχει το αρχείο `src.txt` στον κατάλογο, παίρνουμε το

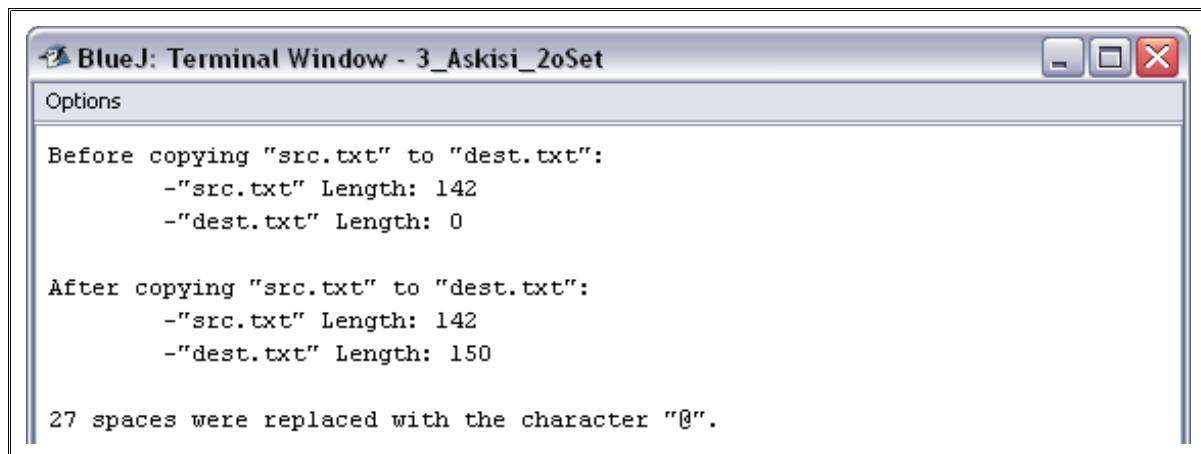
παρακάτω αποτέλεσμα:



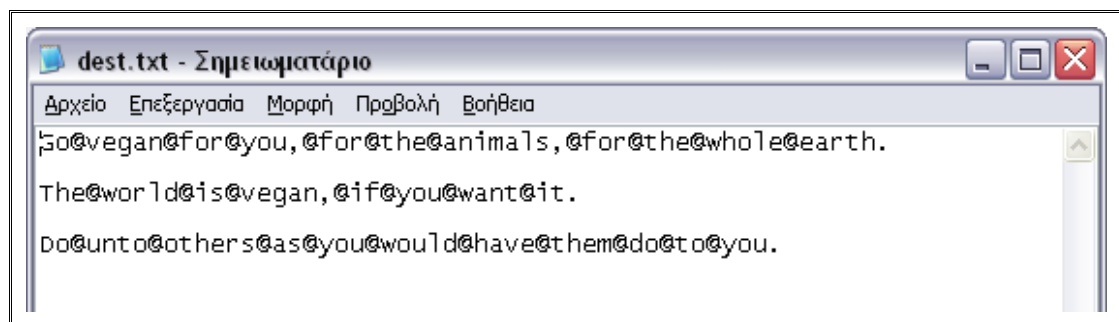
Δημιουργούμε τώρα το αρχείο `src.txt` με το παρακάτω περιεχόμενο:



Έπειτα από την εκτέλεση του προγράμματος, παίρνουμε τα εξής αποτελέσματα:



Κοιτώντας το αρχείο `dest.txt`, παρατηρούμε πως όντως έχουν γίνει οι αλλαγές που θέλουμε:



Έχουν, δηλαδή, αντικατασταθεί τα κενά με το χαρακτήρα «@», ενώ για κάθε γραμμή, προστέθηκε μία κενή γραμμή αμέσως μετά, ακριβώς όπως μας ζητείται στην εκφώνηση. Το μόνο πρόβλημα που υπάρχει σχετικά με την ορθή λειτουργία του προγράμματος είναι πως στην τελευταία γραμμή, ουσιαστικά στο τέλος του εγγράφου, έχει προστεθεί και μία επιπλέον κενή γραμμή, αυξάνοντας έτσι και το μήκος του `dest.txt` κατά 2 byte.