DIFFICULTIES WITH COLLECTION CLASSES IN JAVA
The Case of the ArrayList Collection

Stelios Xinogalos
Department of Technology Management, University of Macedonia, Loggou-Tourpali,59200 Naousa, Greece

Keywords: Teaching Object-oriented Programming, Object Collections, ArrayList.

Abstract: This paper describes research on teaching Object-Oriented Programming (OOP) concepts to undergraduate students. The research focuses on the difficulties of using collections for grouping objects, which is a very common task in object-oriented applications. This research was motivated by the observation that ArrayList collections are a source of various difficulties, combined with the fact that these difficulties have not been investigated before in the literature. The data analyzed come from an undergraduate course on "Object-Oriented Design and Programming", which uses Java and the educational IDE BlueJ. The research carried out used both a qualitative and a quantitative research method. In this paper we present an analysis of the difficulties regarding ArrayLists that aims at providing a useful resource for those teaching OOP.

1 INTRODUCTION

Teaching and learning OOP concepts is accompanied with many difficulties. Many researchers identified this fact and extended research has been carried out regarding various aspects of teaching and learning OOP. This research focuses on difficulties and misconceptions about the fundamental concepts taught first in an OOP course:

- Declaring and calling (multiple) constructors (Carter & Fowler, 1998) and understanding their real essence (Fleury, 2000).
- Confusing instance variables (attributes) with local variables (Truong, Roe & Bancroft, 2004) and shadowing of instance variables by local variables.
- Defining and calling void and non-void methods (Hristova, 2003).
- Distinguishing between classes and objects (Carter & Fowler, 1998; Holland, Griffiths & Woodman, 1997).
- Distinguishing between objects and instance variables, the identity and attributes of an object, an object and a simple record (Holland, Griffiths & Woodman, 1997), and in general misconceptions regarding the fundamental notion of object.

More advanced, but still fundamental for OOP, concepts such as inheritance, abstract classes and interfaces have not been examined so thoroughly, while others, such as collections for grouping objects have not been examined at all. Our belief that it is about time to start studying students' difficulties in concepts like that, led us in undertaking research on Java collections for grouping objects. One of the most popular Java collections is the `ArrayList` collection. The advantages of `ArrayList` collections are many and researchers state that it should be introduced first and emphasized over Arrays (Jacobson & Thornton, 2004; Ventura, Egert & Decker, 2004).

This paper focuses on the difficulties of using ArrayList collections for grouping objects, which is a very common task, even in small scale programs developed by undergraduate students. First, we present the research design and method and then an analysis of the results. Finally, we present the conclusions and future plans for research.

2 RESEARCH DESIGN-METHOD

2.1 Research Rationale & Motivation

The research regarding ArrayLists presented in this paper is part of a long-term assessment of an OOP course that aims at teaching the fundamental concepts of OO design and programming to undergraduate students. The course uses Java, the BlueJ environment (Kölling, Quig, Patterson &

Xinogalos S. (2010).
DIFFICULTIES WITH COLLECTION CLASSES IN JAVA - The Case of the ArrayList Collection.
In *Proceedings of the 2nd International Conference on Computer Supported Education*, pages 120-125

Rosenberg, 2003) and the accompanying book (Barnes & Kölling, 2004), and is highly supported by the use of computers. Specifically: (1) the course comprises of a weekly 2-hour lecture and lab where students use computers for developing programs; (2) educational material is available through an asynchronous course-management, tele-education platform, called CoMPUs; (3) problems that require the development of computer programs are assigned and submitted on a weekly basis through CoMPUs; (4) students are encouraged to utilize the "conversation area" of CoMPUs for interaction with each other and the instructor regarding difficulties with the taught concepts and the assignments.

The first results of the course we taught showed that students face many difficulties with ArrayLists. Since we could not locate any research regarding ArrayLists in the literature, our efforts focused on recording students' difficulties and, if possible, forming categories that would help us and other educators both in designing more effective didactic situations and carrying out further research on ArrayLists. The research carried out the first year used, mainly, a qualitative research method, and data was collected by observations, informal interviews, keeping notes at labs, analyzing students programs, and formal exams in the middle and at the end of the semester. The analysis of the vast amount of data from the 1st course showed that students faced many difficulties with ArrayLists (Xinogalos, Satratzemi & Dagdilelis, 2006). This fact motivated us in carrying out further research on the topic.

2.2 Research Questions

Question 1: Which are the main difficulties regarding the definition and manipulation of ArrayList objects?
Question 2: What are the key concepts of OOP that students must have mastered prior to teaching object collections?

2.3 Research Design & Method

In the next sections we present the results of a written exam, participating in the final grade of the course, that took place in the middle of the semester the 2nd year of teaching the course.

Students were given the skeleton of: (1) a `Candidate` class, used for representing candidates for the "Cambridge First Certificate" or "Certificate of Proficiency"; and (2) an `ExaminationBook` class, used for grouping the candidates of the two certificates in two distinct `ArrayList` fields (called `FCE` and `PCE`). The `Candidate` class had three fields called `name`, `title` and `money` for storing the name of the candidate, the title of the certificate and the amount of money paid respectively, as well as accessor and mutator methods. In the `ExaminationBook` class students were asked to implement various methods that required manipulating ArrayLists.

3 ANALYSIS OF RESULTS

3.1 The Results

Next, we present the results of the study in a tabular form. In all cases, the percentages presented in the "Correct" and "Errors" columns are calculated based on the number of students that actually answered the corresponding question and not on the 64 students that participated in the exam. We must mention that students used Java 4.0 at that time, which means that objects stored in an `ArrayList` were treated as `Object` type and casting had to be used. In the excerpts of code presented bold face is used for marking error-prone code elements.

3.1.1 Defining Accessor Methods

Defining a *get* method is quite straightforward. However, students face difficulties that are presented below. In the case of *get* methods we present the results both for "`String`/`int`" *get* methods (Table 1) and "`ArrayList`" methods (Table 2), since the results are significantly different.

Table 1: "String" and "int" get methods.

Categories of students' replies	%
Correct	72
No answer	16
Errors with the Return type	19
▪ `int` (instead of `String` in 2 methods)	9
▪ void	6
▪ `String` (instead of `int` in 1 method)	2
▪ return type is missing	2
Errors with the Return statement	
▪ wrong values are returned (usually the	6
▪ return statement is missing	2
▪ <field>.get();	2

Table 2: "*ArrayList" get* methods.

Categories of students' replies	%
Correct	41
No answer	28
Errors with Return type	48
▪ String	28
▪ Int	11
▪ Void	4
▪ Candidate (the type of the objects stored in the	2
▪ return type is missing	2
Errors with Return statement	13
▪ the size of the ArrayList is returned	9
Iterates the ArrayList and prints the objects retrieved	4

3.1.2 Adding Objects to an ArrayList

Students were asked to implement an `addCandidate` method with a `Candidate` object as a parameter that adds the object in the appropriate `ArrayList` field according to the value of the object's `title` field. Next, we present the definition of the `addCandidate` method and the results from students' answers (Table 3).

```
public void addCandidate(Candidate
                              aCandidate)
{
   String temp = aCandidate.getTitle();
   if ( temp.equals(
         "Cambridge First Certificate") )
      FCE.add(aCandidate);
   else if ( temp.equals(
      "Cambridge Proficiency Certificate") )
      PCE.add(aCandidate);
   else
      System.out.println("There is no such
                              certificate");
}
```

Table 3: Adding objects to an ArrayList.

Categories of students' replies	%
Correct	14
No answer	31
Errors in Return type	7
▪ return type is missing	2
▪ wrong type	4
Errors in Parameters	20
▪ wrong parameter type: String	16
▪ parameter is missing	2
▪ String FCE, String PCE	2
Errors in accessing private fields outside their class	
▪ direct access of a private field outside its class	41
▪ accessing the instance aCandidate instead of its	16
▪ direct access of a private field outside its class	5
Use of = = or = instead of equals	61
Adding objects to an ArrayList	23
▪ FCE += Candidate;	5
▪ FCE++;	5

3.1.3 Iterating and Printing an ArrayList

Students were also asked to implement a method for iterating the `FCE ArrayList` field and printing the name of all the candidates that have paid (the `money` field has a value greater than 0), the number of candidates that have paid and the amount of money collected. A typical implementation of this method is presented, as well as a review of students' answers (Table 4).

```
public void showFCECandidates()
{
   int num = 0;
   int sum = 0;
   Iterator it = FCE.iterator();
   while ( it.hasNext() )
   {
        Candidate aCandidate = (Candidate)
                                  it.next();
       if ( aCandidate.getMoney() > 0 )
       {
          System.out.println(
                     aCandidate.getName());
          num++;
          sum += aCandidate.getMoney();
       }
   }
   System.out.println("Candidates Number:"
                                + num);
   System.out.println("Total: " + sum);
}
```

Table 4: Iterating and printing an ArrayList.

Categories of students' replies	%
Correct	8
No answer	41
Errors in Return type	13
▪ return type is missing	5
▪ wrong return type is used (String, int,	8
Parameter is used (without reason)	18
Errors in accessing private fields outside their	
▪ direct access of a private field outside its	47
▪ direct access of a private field outside its	3
Errors in retrieving objects from an ArrayList	45
▪ a while loop is used but objects are not	13
▪ the name of the ArrayList field is used as	13
▪ the ArrayList is not iterated	8
▪ the retrieved object (it.next()) is not	5

3.1.4 Creating and Returning an ArrayList as a Subset of an Existing One

A method that iterates an `ArrayList` object passed through a parameter to it, creates and returns a new `ArrayList` containing the candidates that have not paid was the next task. A sample implementation and students' errors (Table 5) are presented.

```
public ArrayList rejectedCandidates(
```

```
                                ArrayList aList)
{
   ArrayList newList = new ArrayList();
   Iterator it = aList.iterator();
   while ( it.hasNext() )
   {
      Candidate aCandidate = (Candidate)
                                    it.next();
      if ( aCandidate.getMoney() == 0 )
            newList.add(aCandidate);
   }
   return newList;
}
```

Table 5: Defining the rejectedCandidates method.

Categories of students' replies	%
Correct	14
No answer	42
Errors in return type	35
▪ void	6
▪ return type is missing	4
▪ wrong return type is used (String,Candidate,	10
Errors in declaring parameters	27
▪ wrong parameter type: Candidate	11
▪ wrong parameter type: int	4
▪ parameter is missing	2
Errors in the use of the ArrayList parameter	13
▪ the ArrayList parameter is instantiated in the	8
▪ all the methods are called for the parameter	5
Direct access of a private field outside its class	8
Errors in copying objects from one ArrayList to	19
▪ wrong argument	8
▪ the argument of the add method is the	5
Errors in retrieving objects from an ArrayList	12
▪ a while loop is used but objects are not	11
▪ the retrieved object (it.next()) is not	5
Errors in return statement	
▪ the ArrayList parameter is returned instead	5
▪ the ArrayList is iterated and its objects are	3
▪ return FCE.size(); return PCE.size();	3
▪ return is used inside the body of while	3

3.2 Review of Difficulties

In this section we review students' difficulties with manipulating ArrayLists based on the results recorded and presented in Tables 1 - 5. Students' difficulties are grouped in 6 categories.

3.2.1 Return Type

The percentage of students that use a wrong return type is much smaller in `void` methods (Table 3: 7%, Table 4: 13%) and methods that return a value of some primitive type or a type considered by students as primitive – such as `String` – (19%, (Xinogalos, Satratzemi & Dagdilelis, 2007)) than the percentage of students that uses a wrong return type in methods that return an `ArrayList` object (Table 3: 48%, Table 5: 35%) or a value of some object type in general. The most common errors referring to return types are the following:

- When a non-void return type is used in a void method, usually *the type of the object or an object's field that is referenced in this method is u*sed (Table 4: 8%).
- *The same percentage of students that uses a non-void return type in a* `void` *method uses* `void` *as the return-type in a non-void method*. For example, see the methods `showFCECandidates` (Table 4: 8%) and `rejectedCandidates` (Table 5: 10%).
- *The return type is missing in a small number of students' programs* (Tables 1-5: 2% - 5%).

3.2.2 Return Statement

In methods were an `ArrayList` object should be returned, some students return the size of the `ArrayList` instead (Table 2: 9%, Table 5: 3%). Also, a small number of students iterate the `ArrayList` that should be returned and prints (Table 2: 4%) or returns (Table 5: 3%) its objects one-by-one.

3.2.3 Parameters

Difficulties with declaring methods' parameters are quite often and some times related to errors with return types, as mentioned above and in (Xinogalos, Satratzemi & Dagdilelis, 2006). The most common errors are the following:

- *A parameter is used without needed*. This error was rare (4%) in simple *get* methods like those of the `Candidate` class (Xinogalos, Satratzemi & Dagdilelis, 2007), but it was much more often in more complicated methods that manipulate objects, such as `showFCECandidates` (Table 4: 18%). In this method most of the students that declared a parameter (13% out of 18%) used an `int` parameter called somewhat "money" that seems to represent the field `money` that has to be tested for every object in the `ArrayList`.
- *A wrong parameter type is used*. However, the parameter type, in *most* cases, is not selected randomly. In the `addCandidate` method that "checks a `String` field of a `Candidate` object in order to add it to the appropriate `ArrayList`", most of the students use a `String` parameter (Table 3). In the

`rejectedCandidates` method where the task is to "add some `Candidate` objects from one `ArrayList` to another", most of the students use a `Candidate` parameter (Table 5).

3.2.4 Accessing a Private Field Outside its Class

Manipulating ArrayLists involves retrieving the objects stored in them and accessing their fields, which should be declared `private`. Although this is a common task half the students fail to access `private` fields correctly outside their class. The most common errors are:

- *Direct access of a private field and without an instance* (Table 3: 41%, Table 4: 47%). This behavior might be a generalization of the fact that students spend most of the time defining methods that access directly the fields that are defined in the same class. Also, this might be a result of students' inability to understand that more than one object of a given class might exist the same time and it is not enough to mention just the name of the field we want to access.
- *Accessing an instance instead of its field* (Table 3: 16%).
- *Direct access of a private field outside its class* (Table 3: 5%, Table 4: 3%).

3.2.5 Retrieving Objects from an ArrayList

Iterating and retrieving the objects stored in an `ArrayList` is a typical task in applications based on such lists. Most textbooks provide templates, but students do not apply them correctly due to a flawed understanding of the `ArrayList` concept. Students use an `Iterator` object and a `while` loop for iterating an `ArrayList`, but the following errors are made:

- *A while loop is used but objects are not retrieved* (Table 4: 13%, Table 5: 11%).
- *The name of the `ArrayList` field is used as type of the variable and casting* (Table 4: 13%).
- *The ArrayList is not iterated* (Table 4: 8%).
- *The retrieved object is not assigned to a variable* (Table 4 & 5: 5%).

3.2.6 Adding Objects to an ArrayList

The `ArrayList` class provides an `add` method for adding objects to it. However, students do not always make use of this method or do not use it correctly (Table 3: 23%, Table 5: 19%). For example, 1 out of 10 of the students used a statement like:

```
FCE += Candidate; or FCE++;
```

for adding a `Candidate` object to an `ArrayList` called `FCE`. Also, several students (Table 5:19%) use a wrong argument in the `add` method.

4 CONCLUSIONS

Manipulating ArrayLists is a skill that all students leaning OOP must acquire. However, this does not seem to be so easy. The difficulties that were recorded in the first teaching of an OOP course (Xinogalos, Satratzemi & Dagdilelis, 2006) resulted in a re-designed course (Xinogalos, Satratzemi & Dagdilelis, 2007) where two lessons were devoted to ArrayLists and lab exercises specially designed to face these difficulties were used. The re-designed course gave definitely better results, according to observation during lab sessions, informal interviews and homework assignments. However, the results of the middle term exams showed that students still face many difficulties. The main difficulties (Research question 1) are:

D1. Students use a wrong return type in methods where an ArrayList object is returned (Tables 2 & 5). Usually, the type of one of the fields of the objects stored in the ArrayList is used as return type. In most cases this is a primitive type or a type considered by students as primitive (such as String).

D2. Students use wrong parameter types in methods manipulating ArrayLists (Tables 3 & 5). In most cases, the type of the parameter is related to the type of the entity (i.e. type of the field, object) being processed in the method.

D3. When iterating an ArrayList students access private fields of the objects retrieved directly, without referring to an instance, or without using accessor methods (Tables 3, 4 & 5).

D4. The size of the ArrayList object or the objects stored in it are returned one-by-one instead of the ArrayList object (Table 5).

D5. Students face difficulties in applying correctly the code pattern for iterating an ArrayList and retrieving objects (Tables 4 & 5).

D6. The add method of the ArrayList is not used, or is used incorrectly (Tables 3 & 5).

D7. Generally, students find it difficult to manipulate a class with fields/attributes of ArrayList type.

The results of the study make clear that many of the recorded difficulties are not related to ArrayLists

in particular, but in key OOP concepts (Research question 2) that students must have mastered prior to their exposure to object collections. These key concepts and the associated difficulties regarding ArrayLists reviewed above, are presented in Table 6.

Table 6: Association between OOP key concepts and ArrayList-related difficulties.

OOP key concept	ArrayList
Object types and not just primitive types	D1, D2,
Classes can have fields/attributes of some	D7
Access modifiers	D3
Accessor methods and especially their	D3
Internal and external method calls	D3

Concluding, further research should be carried out regarding the effective teaching and learning of ArrayLists, and generally object collections. This research could focus on: (1) using the findings of our research for devising and evaluating more effective teachings; (2) exploiting the method of our research for conducting similar studies in order to validate the results; (3) examining in what degree BlueJ supports the comprehension and use of ArrayLists, as well as examining if other environments provide greater support in this issue; (4) devising, if necessary, new tools (educational software) for supporting students in comprehending object collections.

REFERENCES

Barnes, D. & Kölling, M., 2004. *Objects First with Java: A practical introduction using BlueJ*, Prentice Hall.

Carter, J. & Fowler, A., 1998. Object Oriented Students?, *SIGCSE Bulletin, Vol. 28, No. 3*, 271.

Fleury, A. E., 2000. Programming in Java: student-constructed rules, *ACM SIGCSE Bulletin, Vol. 32, Issue 1*, 197-201.

Holland, S. Griffiths, R. & Woodman, M., 1997. Avoiding object misconceptions, *ACM SIGCSE Bulletin, Vol. 29, No. 1*, 131-134.

Hristova, M., Misra, A., Rutter, M. & Mercuri, R., 2003. Identifying and Correcting Java Programming Errors for Introductory Computer Science Students, *ACM SIGCSE Bulletin, Vol. 35, Number 1*, 153-156.

Jacobson, N. & Thornton, A., 2004. It is Time to Emphasize ArrayLists over Arrays in Java-Based First Programming Courses, *ACM SIGCSE Bulletin, Vol. 36*, Number 4, 88-92.

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J., 2003. The BlueJ system and its pedagogy, *Computer Science Education, 13(4)*, 249-268.

Truong, N., Roe, P. & Bancroft, P., 2004. Static Analysis of Students Java Programs, *6th Australian Computing Education Conference*, 317-325.

Ventura, P., Egert, C. & Decker, A., 2004. Ancestor Worship in CS1: On the Primacy of Arrays, *OOPSLA '04*, 68-72.

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2006), Studying Students' Difficulties in an OOP Course Based on BlueJ, *9th IASTED International Conference on Computers and Advanced Technology in Education*, 82-87.

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2007), Re-designing an OOP course based on BlueJ, *7th IEEE ICALT Conference*, 660-664.