MSc in Advanced Computer Science

"A scalable analysis of certain basic Java constructs mistakes, as coded from novice programming students and extracted by means of the Blackbox data set."

Submitted by: Andreas Nikolakopoulos

Date: 12th September 2016

Supervisor: Prof. Ian Utting

E-mail: an310@kent.ac.uk

MSc in Advanced Computer Science

CO880: PROJECT AND DISSERTATION

'A scalable analysis of certain basic Java constructs mistakes, as coded from novice programming students and extracted by means of the Blackbox data set.'

Submitted by: Andreas Nikolakopoulos

Date: 12th September 2016

Supervisor: Prof. Ian Utting

Word-Count: 11.619 words

ABSTRACT

In 2013, with the release of BlueJ's Blackbox data collection project, the computing education research community and particularly the researchers in the field of beginners programming pedagogy, had acquired an important tool to facilitate their analyses on early programming interaction; a massive data repository comprised of hundreds of thousands of worldwide beginner-users Java programming-code (an ample sample of research subjects') – which aided them to circumvent the toil of data-collection, and simultaneously to generalise promptly and unbiasedly their results.

On that same basis, this project retrieves a robust sample of 'Runtime Errors' within Blackbox for a time-frame period of twenty-six months, and subsequently assesses three basic Java constructs' (i.e., 'Arrays', 'Strings' and 'Collections'), which have been identified to cause most of the novices' runtime errors. Ultimately, it reports its outcomes pertinent to: the novices interactions with these constructs; the instructors 'Experts blind spots' (mistaken viewpoints about the commonest novices pitfalls with these constructs), and certain previous smaller-scale studies' conclusory allegations on these constructs (which till then because of those studies meager subjects' sample sizes could only hypothetically be acknowledged as reliable).

This project's uppermost target (i.e., the composition of a framework to exhibit clearer the role of certain particularly counterintuitive Java constructs as applied to beginner students learning) – had been eventually attained.

ACKNOWLEDGMENTS

I would like to express my appreciation to all those who have supported me throughout the course of this project; and particularly, my supervisor Professor Ian Utting for his always motivating stance and for his valuable remarks and guidance; Dr. Neil Brown for his aid in circumventing the impediments arising with Blackbox; Professor Sally Fincher for her clarifications pertinently to certain literature studies queries; and finally, all the rest of the university's Computing Education Group, my family and my friends, whose suggestions and encouragement have contributed significantly towards the more methodical and coherent completion of this study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This project's aim has been, to retrieve initially from a massive dataset of novice users Java source-code (BlueJ's Blackbox) a significant amount of their runtime errors, and subsequently to extract the most problematic constructs' within them – synthesising thus (due to the large participants' sample) a 'concrete/unbiased framework of the learners commonest Java construct pitfalls'; in other words, *the composition of a basis exhibiting clearer the behavioural traits of beginners interactions with certain of their most troublesome constructs*. The idea had emerged from the fact that heretofore (before the appearance of Blackbox), there had been many smaller-scale studies on Java constructs comprised of 'grey-areas' presumptions, principally attributable to those researches' meagre participant-subjects' sample sizes; smaller-scale studies, which lacked a solid benchmark constructs' context (i.e., the framework) over which to juxtapose their potentially-biased findings, and validate whether their smaller-scale results could be reliably generalized, or not.

Hence, targeting Java's 'Runtime Errors' domain incorporated inside Blackbox, and subsequently retrieving for a period of twenty-six months numerous runtime exception data, three Java constructs (i.e., 'Arrays', 'Strings' and 'Collections') were identified to constitute the principal culprits for most of the novices' runtime error instances; three constructs, which had been further meticulously examined, with a particular emphasis placed on the most problematic construct of 'Arrays'. The results of this endeavour had revealed, that despite some few limitations in interpreting Blackbox's bulk data (predominantly due to the scarce knowledge about its participant users), BlueJ's Blackbox data collection project, had overall greatly supported the composition of this study's framework of the novices most fallible Java constructs; considering that the voluminous data acquired, had permitted with much more assurance, to deduce new interesting insights about the novices interactions with their most problematic constructs; to dissolve certain instructors 'Experts blind spots' (mistaken beliefs in what truly constitute common novices constructs bugs); and to ratify the broader applicability of certain previous smaller-scale literature studies' conclusions on novices constructs' misconceptions (which up until then could be only hypothetically acknowledged as non-biased and concrete).

Thus more particularly, to elucidate briefly upon what each of the following Chapters will examine:

Chapter 2 provides a rudimentary overview of the BlueJ BlackBox project, and of the programming languages constructs' context; along with a synoptic description of the process that this study underwent, before reaching the decisions: i) to investigate the role of 'Java constructs' as applied to novices learning, and ii) to focus on Java's 'Runtime errors' within BlackBox (as the source from which to extract the novices buggiest constructs). Chapter 3, presents the primary retrievals/classifications of the novices 'Runtime errors' data located within Blackbox; in conjunction with an evaluation of the most problematic Java constructs (emanating within those voluminous errors data), and the consequent first essential insights with respect to the beginners' interactions (misconceptions) with these Java constructs. While Chapter 4, is composed of a report, whose objective has been to validate the reliability of a significant part of this study's exacerbated preliminary findings of Chapter 3; and specifically that 'Arrays' (associated with the ArrayIndexOutOfBoundsException class) – identified as the Java construct within Blackbox liable for generating the most novices 'Runtime errors' –

indeed constituted a predominantly challenging beginners theme, even after a juxtaposition of this assertion against several analogous past literature studies.

In its turn Chapter 5, exhibits several further insights about the beginners' interactions with the construct of 'arrays', following a final supplementary and more laborious inquiry on BlackBox's 'runtime exception errors' (unfortunately confined solely to 'arrays' due to time-limitations); an inquiry, which this time has looked into a fair sample of the novices actual programming source-code exception cases ('actual `.java` source file *contents*'), to identify the array's most defective constituent sub-constructs' categories conducting novices to most ArrayIndexOutOfBoundsException errors. Whilst, lastly Chapter 6, offers an account of the experiences that were encountered during the course of this research project; i.e., the occasional impediments arising from Blackbox's bulk data's scarce knowledge about its participant users, along with the various broad-spectrum/enlightening insights stemming from the preceding Chapters most significant outputs; in addition to, some proposals for those researchers interested in further progressing this project's investigation (i.e., enhancing the framework of the novice learners most common [problematic] Java constructs').

2. BLACKBOX SERVER & JAVA CONSTRUCTS

This Chapter provides an overview about the BlackBox data collection project and the programming language constructs' topics; and additionally, describes how this project's ideas, to investigate the programmer-users early interactions with Java constructs' and to focus on the novice learners 'Runtime Errors' domain within Blackbox (for the subsequent extraction and assessment of the beginners commonest and gravest constructs' pitfalls), had been both initially conceived.

Hence, to begin with, considering Guzdial's (2015, p.3) view that the greatest challenge of computing education research is profoundly in understanding how students learn programming, and acknowledging that in programming education there are still many associated grey areas mutually applicable to both the teaching of programming as well as in the design of educational tools for programming (Guzdial 2015, p.3; Kölling 2012) − it would be reasonable for someone to suggest that the existence of a large database comprised of novice students source-code, could prove very useful towards the apprehension/clarification of these grey areas; a large database like that, accessible to any researcher in the computing education community, could facilitate researchers, as it would allow them to assess promptly if their findings could be reliably generalised; and furthermore to achieve that, by circumventing the toil of data collection (i.e., collecting their students interaction data with the computer system, evaluating their students assessments/grades, performing surveys and constructing subjective-rating questionnaires). Unfortunately heretofore, the norm of most literature studies concerning the research of students early programming interaction, had most of the times, been based on the collection of a small data-sample of programming-code taken for each case study respectively, from a limited number of beginner students of a specific institution's CS1 module; something, which had ultimately rendered these studies' results as doubtful (because of their participants meager sample size), and which had furthermore prohibited the chance of their sample's data reuse (as their retrieved sample's data had been subsequently typically discarded after the end of these case-studies). It had been under this context that the idea for the Blackbox project had emerged.

BlueJ is a pedagogical Integrated Development Environment (IDE) specifically designed for introductory teaching. It is being used mainly by first-year course students of universities who are learning object-oriented programming in Java, and since its first release in 1999, it has nowadays expanded to a worldwide scale, encompassing more than 1.8 million active users every year. On June of 2013, upon the release of BlueJ's 3.1.0 version, the Blackbox data collection research project, which was incorporated as a plug-in to the BlueJ 3.1.0 IDE, was concurrently launched. BlueJ's Blackbox is epigrammatically an ongoing data collection research project, which collects anonymously large amounts of data from BlueJ users (mostly beginner students), who have voluntarily opted-in to participate in this research; it records their IDE interactions and their source code edits (e.g., which methods are invoked and when, what errors are encountered and how frequently) in a MySQL server hosted at the University of Kent, with the goal to make this large database available to other research groups wishing to conduct studies and increase their understanding upon the way beginner students learn programming; benefiting thus, the wider computing education research community.

In mid-year of 2015, during the time of this project's goal conception, a review was made to all publications relevant to the Blackbox project, which by that time, given Blackbox's still novel manifestation, had been mostly encompassed from a considerable number of reports and studies of a principally informative and expository character (Brown et al. 2014; Brown 2013; Kölling 2012; Stevens and McCall 2012; Utting et al. 2012); among those studies one such introductory paper (Brown et al. 2014) had stood out, since among others, after having presented some examples of the sort of analyses that interested researchers could perform in using Blackbox's collected data, it had additionally proposed some suggestions of the kinds of topics that could be addressed through the source code's analysis. Consequently, following this paper's exhortation (Brown et al. 2014, p.227), the topic of programming *constructs* had been selected for further investigation; and that after acknowledging from the literature their double-facet nature, first to serve as the novice programmers building blocks towards the development of solution plans in programs (Ebrahimi 1994, p.475), while meanwhile secondly, to prohibit learning as a frequent source of misconceptions for novice programmers (Kuittinen and Sajaniemi 2004, p.57; Ebrahimi 1994, p.478)

A language *construct* is a part of a program that can be formed by one or more tokens[1], which if in turn conform to a specific programming language's syntax and semantics rules, can eventually form 'language sentences' (Cprogramming.com 2006). In other words, language *constructs* are the base units from which a programming language is made of; albeit though as noted above, a frequent cause of difficulties for beginner programmers as well. Tew (2010, pp.56-59) in her study had further affirmed that, by demonstrating that while for novice students introductory language constructs (i.e., fundamentals, logical operators, selection statements, definite loops), did not seem to be a factor of confusion, oppositely the appropriate construction of the more sophisticated program constructs (i.e., function/method parameters, function/method return values, and recursion), did indeed prove to be challenging for them to master.

Considering the role of programming language constructs as applied to beginners errors, and particularly by looking at the studies of Soloway, Bonar and Ehrlich (1983), in which novice students found it easier to program correctly through a programming language ('Pascal-L') whose constructs could facilitate certain programming strategies more cognitively preferable to them[2]; but also of Fisler (2014), whose students using a functional programming language's ('Racket') limited set of higher-level programming constructs, had been able to solve better from previous studies one of the most studied problems of computing education research – it had become evident that *the CS1 programming language choice* was a critical factor towards the apprehension of novice students programming behaviour. Many of the novice students problems appeared to be the artifacts of the CS1 programming language's choice; something that Ebrahimi (1994, p.458) seems to have uprightly remarked upon, by claiming that the type and number of errors varied from language to language.

---

[1] i.e., *tokens* are a concatenation of string characters that form elemental units like:

`int , + , ::`	(tokens: *primitive variable*, *symbol*, *operator*)
`int sum_2_numbers (int x, int y)`	(construct of the type *function*)
`class MyClass { /* ..... */ }`	(construct of the type *class*)

[2] (i.e., in the Pascal-L programming language, the inclusion of the construct `leave;` the equivalent of the `break;` construct in Java)

> different programming languages evoke different programming styles, thus generating their own type of errors. As a result, *the type and number of errors vary from language to language* (Knuth 1971; Kleler 1984; Ebrahimi 1989 cited in Ebrahimi 1994, p.458, my italics).

In this project since Blackbox is by 'de-jure' concerned with capturing specifically Java code data through BlueJ, the focus will be exclusively placed upon the Java object-oriented programming language; Java, which among others, through its relatively clean syntax and semantics, and its elimination of certain types of error-prone programming constructs (e.g., pointers, multiple inheritance), has become today, an important component of most undergraduate CS1 introductory courses. The concentration of this project in its entirety to Java, has fortunately rendered it bereft from the requirement of a juxtaposition of the Java language against other popular CS1 introductory programming language choices (Böszörményi 1998; Pont 1998), or along other widely-used commercial programming languages (Batool et al. 2015). Though, nonetheless, the '*programming language choice*' in introductory computer science classes, is not the only factor for which concern should be attributed in regards to its role as a source of students difficulties towards programming-language constructs apprehension; as according to Guzdial (2015, pp.30-33), a second origin from which novice students errors towards programming-language constructs apprehension stem from, is also the higher-education teachers lack of *'experience in the programming language taught*'.

More specifically, according to Guzdial (2015, pp.30-33), in CS1 introductory courses it is often the case, that amidst the teachers effort in helping novices see a program as a whole, understand its main parts and of their relation, what in the CS terminology would be called 'the building of a mental model of a notional machine' (Boulay, 1986, pp.58-59), unfortunately many times teachers fail to look through the novices' eyes; but instead, based upon their own advanced content knowledge, misinterpret what would constitute a novice student's relative difficulty, and succumb under what Nathan, Koedinger and Alibali (2001) have labelled as an '*expert's blind spot*'. Gudzial (2015, p.59) in citing GREEN's (1977) study, has demonstrated an instance of such an 'expert's blind spot'; by exhibiting how upon contrasting two *`if`* 'selection statement' language constructs variations (i.e., the (a) *`if`* without *`else`* case, against the (b) *`if`* with *`else`* case), contrary to most textbooks assumptions, which teach both cases (a) and (b) consecutively, believing that novice students acknowledge them as equally easy, evidence had adversely shown that novice students comprehension response times for case (b), had been ten times slower; and more erroneous in juxtaposition to case (a).

It is the dissolution of such kind of qualitative research incongruities (e.g., higher-education teachers 'expert blind spot' phenomena) that Utting et al., (2012, p.4), had foreseen that the large scale dataset of Blackbox would be able to shed light upon, and which moreover constitute part of this project's targets. In particular, this project's goal, is to retrieve a significant amount of errors data from Blackbox, and subsequently to create *a concrete elaborated framework of certain particularly problematic for novices Java constructs*; so that for example 'expert blind spot' instances, or potentially biased findings of previous literature case-studies[3] (based upon those same Java constructs types, albeit collected from much smaller dataset samples), to have a grounds (i.e., the framework) upon which to be

[3] (e.g., Batool et al. 2015; Tew 2010; Dale 2006)

interpolated along, and validate if their small-scale results upon the novices' misconceptions, could be solidly generalized.

Yet, at this point, considering the vast size of over 500.000 Blackbox users (Brown et al. 2014, p.225), and the large number of captured data options that the Blackbox server offered for investigation, the dilemma was to identify the appropriate dataset tables amidst the large Blackbox MySQL server; those, whose retrieval would provide the most meaningful Java language constructs data (both quantitatively and subsequently qualitatively). Luckily, at this juncture, Professor Ian Utting, this project's supervisor, as also a member of the BlueJ IDE research team and co-implementator of Blackbox, having had meticulously examined all material relevant to the Blackbox project, had suggested targeting the Java language's 'runtime errors' incorporated inside Blackbox; a domain abundant in programming-language constructs data, which by that time[4] still, had been unexploited in terms of a previous large-scale investigation − making this study's expected findings even more compelling.

[4] before the study of Pritchard (2015)

3. Initial Data Findings

This Chapter exhibits certain insights that had been inferred about the beginner-users behavioral patterns with their three most problematic Java constructs: 'Arrays', 'Strings' and 'Collections' (emanating within most of their 'runtime errors'); after previously describing the process that these three constructs underwent (i.e., runtime exceptions data retrieval, subsequent data processing, etc.) before their particularly problematic nature had been ultimately identified as such.

Thus, to begin with, initially an inquiry was made among a number of publications (i.e., relevant electronic and academic sources) for the identification of those 'runtime exception classes', which had up till then, been generally considered as the most problematic ones – expecting them also on that account to hold the most novices constructs errors; an inquiry, which based upon these sources[5], had at its onset exposed 14 'runtime exception classes' as those generally regarded as the most troublesome for users. Consequently, the exception error instances for these 14 'runtime exception classes' had been retrieved from Blackbox for a period of 26 months (Appendix A), to reveal that among these 14 classes, only 10 had an overall percentage-share of error instances larger than 1% – the other 4 had only scarce number of error instances (and were disregarded). In addition, among the remaining 10 'runtime exception classes' (Figure 1), only 5 were found to have an overall percentage-share of error instances larger than 3%; and thus following the 'Cut-Off-Sampling' method's principles, which states that a set of units can be deliberately excluded if its contribution to the total is small (Haziza, Chauvet and Deville 2010, p.303), the set of *smaller 'runtime exception classes'* of 1% and 3% *were cast aside*, and the focus was placed only to the 5 biggest ones[6]; those, whose thresholds accounted for a 91% of the overall Blackbox users runtime exception errors.

[5] (among which indicatively: Beginnersbook.com 2014; Ben-Ari 2011; 2007; www.tutorialspoint.com 2009; Java2s.com 2002)

[6] (ArrayIndexOutOfBoundsException, NullPointerException, StringIndexOutOfBoundsException, IndexOutOfBoundsException and ArithmeticException)

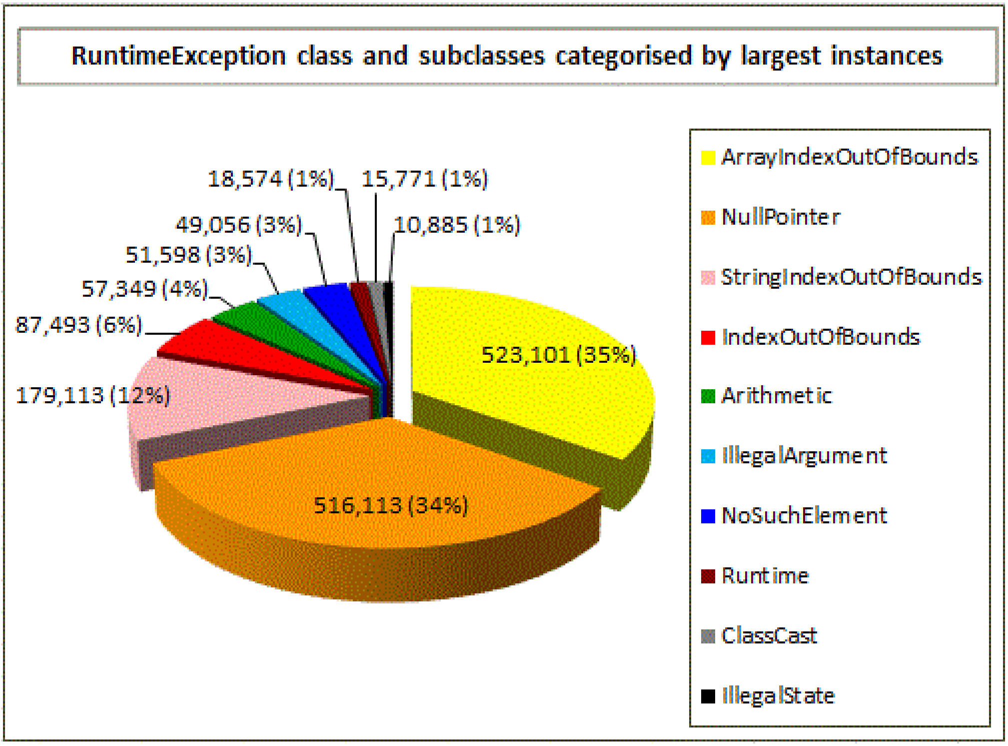


Figure 1 – RuntimeException Class and Subclasses Categorised by Largest Instances

Subsequently, after identifying, retrieving and hierarchically classifying for a period of 26 months the Blackbox users 5 most troublesome '*runtime exception classes*' data (and before proceeding to extract within them the most common [problematic] associated constructs' errors), at this stage it was deemed beforehand prudent for a supplementary, analogous data retrieval/classification to be carried out; and more specifically, to those 5 most buggy 'runtime exception classes' errors – '*exception errors messages*' data (i.e., 'exception error message' in this case: refers to the short descriptive message text following the name of the exception, contained in the first line of information displayed in response to the invalid input [stack-trace] that led to the exception). The idea for this subsequent data analysis, stemmed from the logic that the procurement of these supplementary elaborated data, would provide *more information about the reasons* why these 5 most problematic 'runtime exception-classes' augmented error instances had occurred; and consequently, similarly why the associated numerous constructs' errors (encompassed within those exceptions) had been also generated[7].

Figure 2, exhibits the ArrayIndexOutOfBoundsException class, along with a classification of its exception-messages' categories (ordered hierarchically according to those exception message types having the highest number of error-instances); while below, some such exception-message categories deriving from the first-lines of stack-traces of the class's errors are demonstrated as examples:

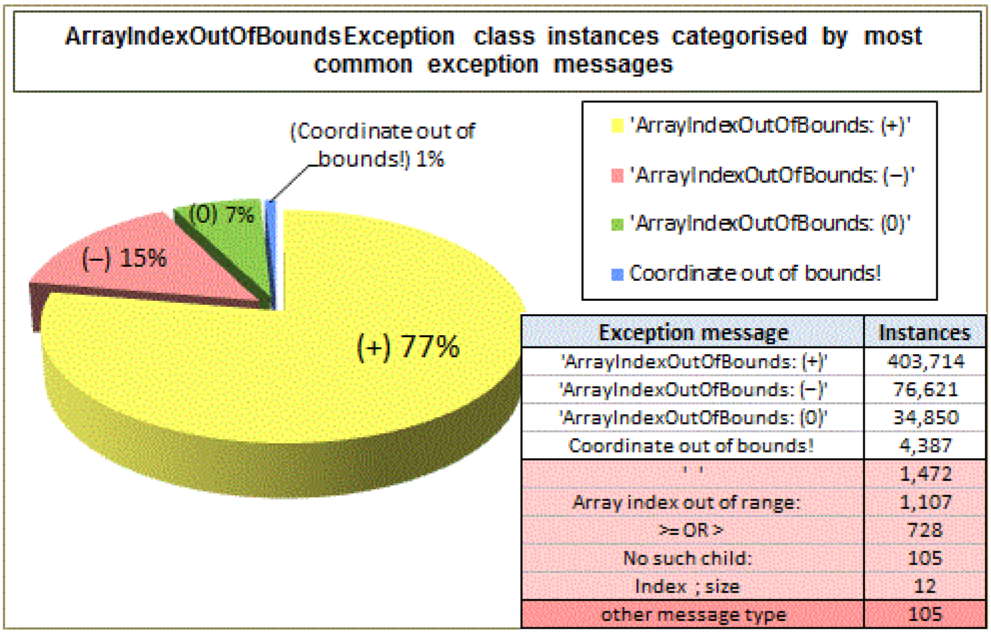
FIRST LINES OF STACK-TRACES FROM ARRAYINDEXOUTOFBOUNDSEXCEPTION ERRORS:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:   4
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:  -2
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:   0
```

[7] (something which eventually had indeed proven useful in the concluding part of Chapter 5.)

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Coordinate out of bounds!
…
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Array index out of range: 1
```

Thus, the extra data (4) above, would be symbolized in Figure 2 under the 'Exception-Message' category 'ArrayIndexOutOfBoundsException: (+)', to indicate those users 'Instances' having difficulty to understand that an array Java program cannot access an integer (int) element with an index value *greater* than or *equal* to its length capacity; the extra data (–2) would be symbolized under the category 'ArrayIndexOutOfBoundsException: (–)' to indicate those users 'Instances' having difficulty to understand that an array Java program cannot access an (int) element with an index value smaller than zero; the extra data (0) would be symbolized under the 'ArrayIndexOutOfBoundsException: (0)' category to indicate those having difficulty to understand that an array Java program cannot access an (int) element with an index *zero* while an array was empty; the category 'Coordinate out of bounds!', those having difficulties performing *image manipulations*, the category 'Array index out of range:', those having difficulties using *vectors*, and so on.



Exception message	Instances
'ArrayIndexOutOfBounds: (+)'	403,714
'ArrayIndexOutOfBounds: (–)'	76,621
'ArrayIndexOutOfBounds: (0)'	34,850
Coordinate out of bounds!	4,387
' '	1,472
Array index out of range:	1,107
>= OR >	728
No such child:	105
Index ; size	12
other message type	105

Figure 2 – ArrayIndexOutOfBoundsException class categorized by its most common exception message types[8] (*exception-message categories which concatenated accounted for a small overall percentage-share of 1% marked as red, were excluded following the 'Cut-Off-Sampling' method's principles).

In continuation, during the course of this project, the interest had further shifted to 3 among the 5 runtime exception classes (i.e., the ArrayIndexOutOfBoundsException, the StringIndexOutOfBoundsException and the IndexOutOfBoundsException classes); considering that these 3 classes could provide a more meaningful insight into 3 basic Java data type constructs' (i.e., 'Strings', 'Arrays' and 'Collections'), which are typically taught in most

[8] In Appendix B the rest of the most problematic runtime exception classes are displayed equivalently to Figure 2, along with a few interpretive supplementary references of all the classes' exception-message categories.

Semester-1 CS introductory-programming first-courses (CS1). Hence, after having initially estimated for the timeframe period of 26 months the 'Total Number' of Instances and the 'Users Number' values for each of the 3 exception classes errors[9], devoid from those users whose lifetime-interaction with the BlueJ IDE had only 1 session (to avoid having a distortion in the results from that dubious hidden parameter of participant-subjects who appeared in the database only once[10]), subsequently, the 'Average-Number of Errors (per-user)' values for each of the 3 classes was calculated (Appendixes C, D, E):

Average Number of Errors (per-user) = Total Number / Users Number

Thereafter, as a starting point in the analysis, the development of all of the 3 runtime exception classes simultaneously were composed under one graph (Figure 3), for a period of 19 weeks (a typical Semester-1 period of an introductory-programming CS1 course); aiming with this to institute a central comparison framework-diagram for the different exception classes 'Average-Number of Errors (per-user)' trends.

Thence, in the graph below, an examination of the possible relationships of the exception classes was performed, and this had revealed that:

- The linear correlation between the ArrayIndexOutOfBoundsException class and the IndexOutOfBoundsException class was (-0.17). That meant that the 'Average-Number of Errors (per-user)' for the two exception classes was overall *negatively* correlated in Semester-1; i.e., while the ArrayIndexOutOfBoundsException class errors increased, the IndexOutOfBoundsException class errors decreased at the start of Semester-1, and vice-versa at the end. Notably, Ventura, Egert and Decker (2004, p.72) in their study, have examined the relationship of these 2 constructs, listing the drawbacks of 'Arrays'; and they have referred to a mistaken typical attitude of educational institutions in introducing students with the more primitive construct of 'Arrays' as the first data-structure[11] early in their objects-first CS1 Semester-1 course[12], instead of the more pedagogically appropriate data-structure topic of 'Collections' − ultimately, proposing their alternation.
- The linear correlation between the ArrayIndexOutOfBoundsException class and the StringIndexOutOfBoundsException class was (0.56). That meant that the 'Average-Number of Errors (per-user)' for the two exception classes had an overall *significantly positive* correlation in Semester-1 ('Arrays' and 'Strings' errors increased and decreased parallelly); this could potentially be due to the fact that 'Strings' and 'Arrays' being both non-primitive object types in which reference semantics apply, have the potential for broadly similar problems in beginning-programmers understanding (Biddle and Tempero 1998, p.49); or perhaps alternatively, because

[9] Using in the MySQL queries of every class, as a filter, those categories of exception message types that were previously indicated to have the highest number of error instances (under appropriate thresholds), and only those which derived from standard Java libraries.

[10] (Jadud and Dorn 2015, p.138; Utting et al. 2012, p.1)

[11] pointed also by Barland (2008, p.259)

[12] designated also from (Ehlert and Schulte 2009, p.20; Howe, Thornton and Weide 2004, p.293)

'Strings' and 'Arrays' are occasionally taught together under one same CS1-session termed as 'Fundamental Data Structures' (i.e., Ehlert and Schulte 2009; 2007; AIJTF 2001, pp.161-175).

- The linear correlation between the IndexOutOfBoundsException class and the StringIndexOutOfBoundsException class was (-0.45). That meant that the 'Average-Number of Errors' (per-user) for the two exception classes had an overall *negative* correlation in Semester-1; i.e., In Figure 3 while StringIndexOutOfBoundsException errors increased, IndexOutOfBoundsException errors decreased for the beginning, and vice versa for the later weeks.

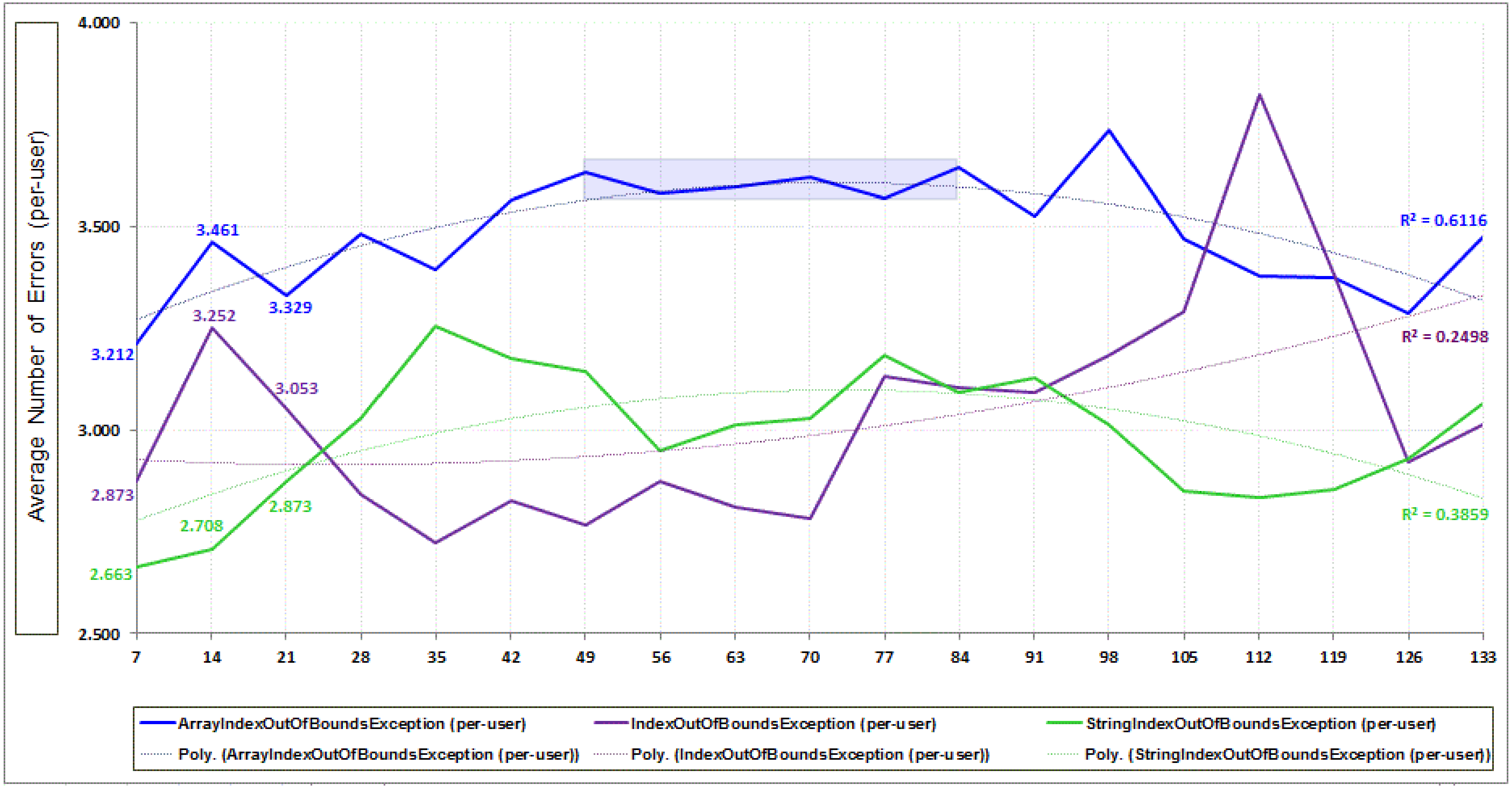


Figure 3 – Comparison of 'Average-Number of Errors (per-user)' between exception classes for a Semester-1 CS1 introductory course over a typical 19 weeks period

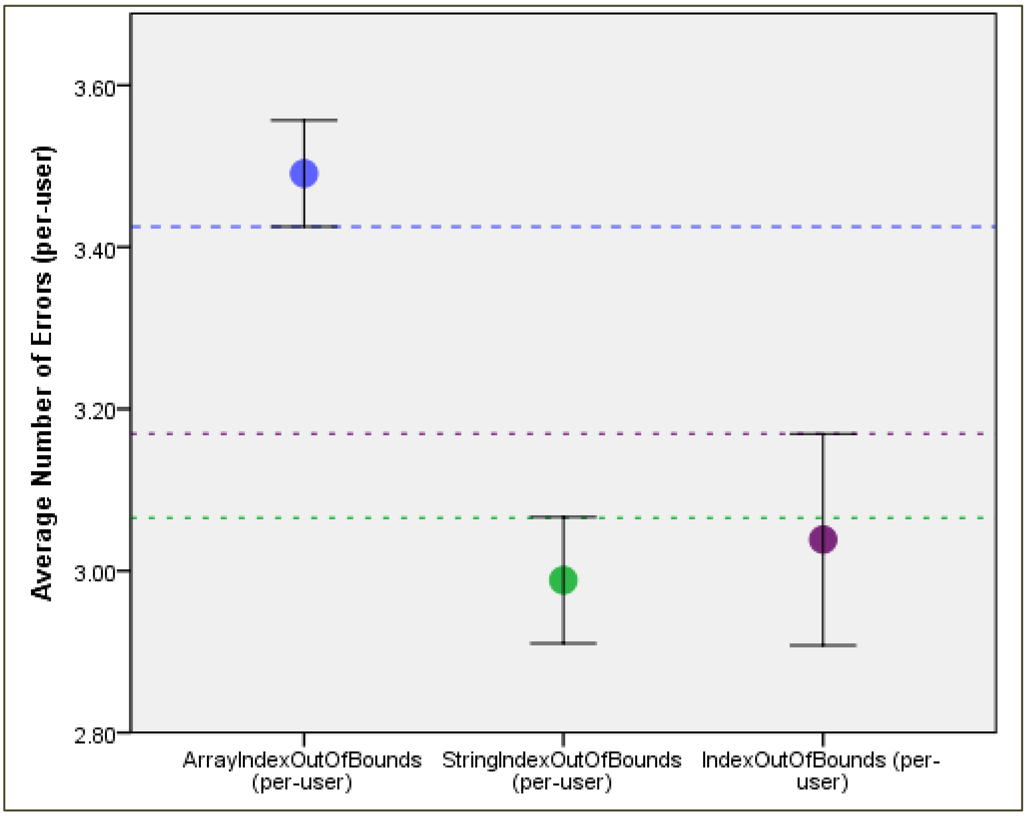


Figure 4 – Distribution of Average Number of Errors (per-user) over Exception classes for Semester-1

Next, wanting to compare visually the 'Average-Number of Errors (per-user)' between the 3 exception classes in Semester-1 via a more apparent comparison, after utilising the sophisticated IBM SPSS Statistics software's Error-Bar graphical tool (with the Confidence-Interval of the corresponding exceptions), Figure 4 was plotted; and several additional findings had emerged:

- Firstly, it was discovered that the ArrayIndexOutOfBoundsException class's 'Average-Number of Errors (per-user)' had been significantly higher in comparison to the other two exception classes at the 95% Confidence Level; whilst concurrently, the difference between the classes of the IndexOutOfBoundsException and the StringIndexOutOfBoundsException, had not been significant. Practically what this meant, was that in Semester-1 *students performed similarly for 'Strings' and 'Collections' in terms of their Average-Number of Errors (per-user)* (2 to 3 weekly errors); while *'Arrays' was considered the most difficult topic* from a learning perspective – significantly surpassing the Average-Number of the other two types of Errors (per-user) (3 to 4 weekly errors).
- Secondly, in summarizing this Error-Bar graph along the trend of Figure 3, it had become obvious upon a casual inspection that in the observed period (Semester-1), the 'Average-Number of Errors (per-user)' for the IndexOutOfBoundsException class had a more unstable behavior in comparison with the other 2 exception classes (a significantly higher variance); something, further illustrated in the SPSS Error-Bar's graph in Figure 4 above, whilst looking at the IndexOutOfBoundsException class's large 95% Confidence-Interval's size-limits. Again, what this practically meant, was that in Semester-1 (19 weeks in Figure 3), *as students proceeded from week-to-*

week, their average number of Collection errors changed significantly (the change from 2.9 to 3.2 on average of Figure 4).

	Semester	Slope (regression coefficients) - point estimates	Std. Error	Sig. (2-tailed)	95% Confidence Interval		Type of Slopes (positive, neutral and negative)
					Lower	Upper	
ArrayIndexOutOfBounds	1	.003	.007	.697	-.012	.018	neutral
StringIndexOutOfBounds	1	.003	.009	.703	-.016	.022	neutral
IndexOutOfBounds	1	.023	.012	.100	-.002	.048	neutral

Table 1 - Distribution of Slope over exception classes for Semester-1[13]

Finally, wanting to examine the 3 exception classes using some well-known statistical concepts, Table 1 was constructed similarly to what Robins, Haden and Garner (2006, p.168) have done in Figure 6 below; and likewise, the distribution of 'Slopes' ('Type of Slopes') had been plotted and named accordingly, so that some useful assumptions could be drawn. The 'Type of Slopes' would practically estimate, if the particular exception classes errors had grown, decreased or had remained constant (around zero) throughout Semester-1; and subsequently if there was a negative, positive or neutral direction towards learning.

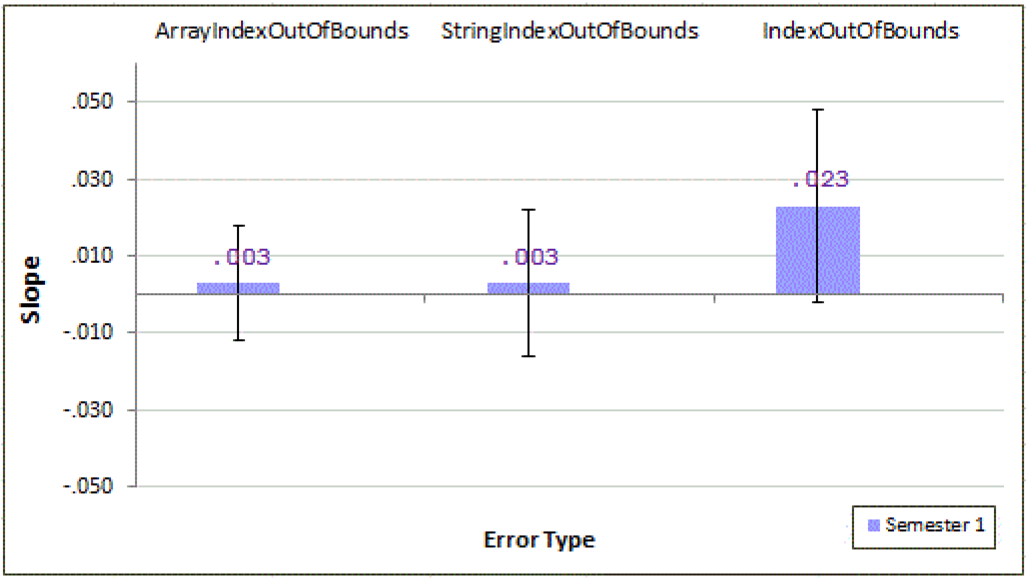


Figure 5 - Slope (regression coefficients) and 95% confidence intervals for Semester-1 of the 3 exception classes

Hence, according to Robins, Haden and Garner (2006, pp.168-170), erroneous programming topics under investigation in Table 1 having 'negative' Type of Slopes, would indicate Java introductory topics whose labs and course content having been successfully assimilated; while conversely, erroneous programming topics under investigation designated as having 'neutral' Type of Slopes, would indicate Java introductory topics being *learned less rapidly*, *in*

[13] TYPE OF SLOPES:

positive: when Slope > 0, and the Sig. < 0.05;

neutral: when Sig. > 0.05; (95% CI includes zero)

negative: when Slope < 0, and the Sig. < 0.05.

need of closer attention (probably being *too difficult* or *developing too quickly*), and with potentially inherently *harder* (or *unevenly distributed*) *consequent labs*.

These designations (Robins, Haden and Garner 2006, pp.168-170), upon a first cross-over inspection between the results of Table 1, and along the views of other scholars, seemed to be in conformance. In particular, Ehlert and Schulte (2009, p.24) in their experiment have found 'Arrays' and 'Strings' as being *too difficult* topics, and partly *developing too quickly*; Reges (2006, p.294) as well as McConnell and Burhans (2002, p. T4G-4) upon their recounted observations pinpointed 'Arrays' as topics *developing too quickly* with *unevenly distributed consequent labs*; and finally Robins, Rountree and Rountree (2003, p.162) have identified 'Arrays' as problematic language features and *in need of closer attention.* However at this point, it should be noted that as with the case of Robins, Haden and Garner (2006, p.170) in examining the Slopes of Figure 5 initially, the difference between positive, negative and neutral 'Type of Slopes' was not obvious upon a casual inspection; it was only after utilising the SPSS Statistics software's 'Linear-Regression' tool (FathomEnthusiast 2011), and testing the significance of the slopes (Types of Slopes)[14] that a categorization in regards to the Slopes of the 3 exception classes for Semester-1, and their respective classification in Table 1 under a 'neutral' Type of Slope, had been feasible.

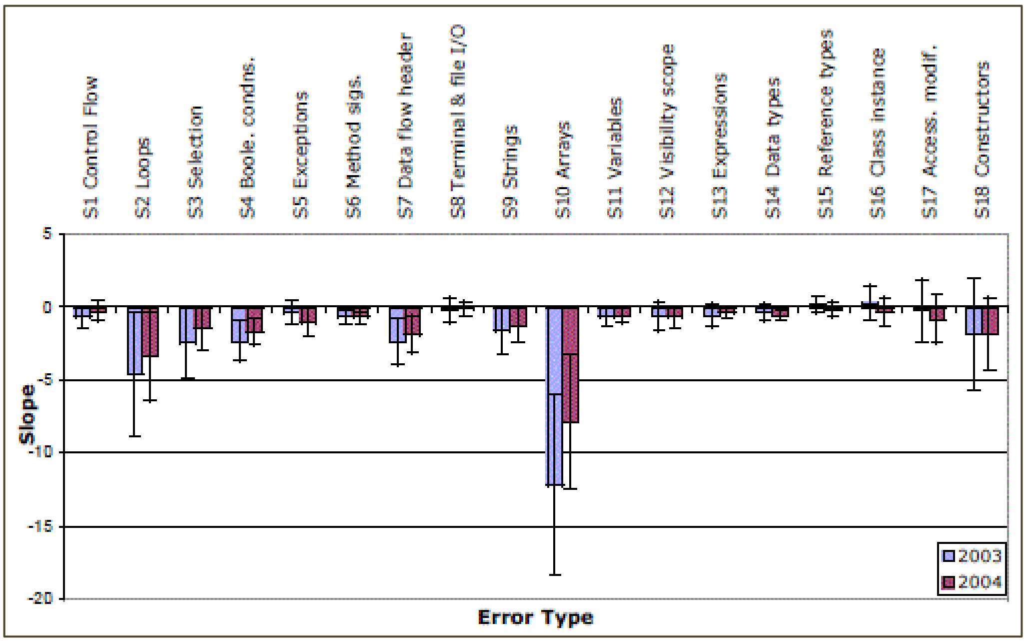


Figure 6 - Slope (regression coefficients) and 95% confidence intervals for 2 years of error types S1 to S18 (Robins, Haden and Garner 2006, p.168)

Eventually at this stage, in assembling all the above findings, some important preliminary outcomes seemed to have already emerged:

- The ArrayIndexOutOfBoundsException and the StringIndexOutOfBoundsException classes had an overall significant *positive correlation* in Semester-1 (i.e., as students

[14] In Table 1 the 95% Lower Bound and Upper Bound Confidence-Intervals included the (0) value – essentially denoting that *the 3 exception classes 'slopes (regression coefficients)' were not significantly different from zero* (Annotated SPSS Output: Regression, 2014).

'Array' errors increased, correspondingly their 'String' errors also increased, and vise-versa). Contrary, the IndexOutOfBoundsException class had an overall *negative correlation* between both the ArrayIndexOutOfBoundsException and the StringIndexOutOfBoundsException classes in Semester-1 (i.e., as students 'Collection' errors increased, their 'Array' and 'String' errors decreased, and vise-versa).

- A highly volatile 'Average-Number of Errors (per-user)' existed for the IndexOutOfBoundsException class. As students proceeded from week-to-week in Semester-1 (19 weeks in Figure 3), *their average number of 'Collection' errors fluctuated considerably* (the change from 2.9 to 3.2 on average of Figure 4); decreasing (suggesting a more positive learning) and increasing (suggesting a more negative learning) unsteadily.
- After constructing the Confidence-Intervals in the Errors-Bar graph of Figure 4, it was found that in Semester-1, while students had about the same 'Average-Number of Errors (per-user)' for 'Strings' and for 'Collections' (2-3 weekly errors), nonetheless, their 'Average-Numbers of Errors (per-user)' for 'Arrays' were significantly higher (3-4 weekly errors). *'Arrays', were by far the students' most difficult topic*.
- The ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException, and IndexOutOfBoundsException classes had been tested in regards to the statistical concept of the significance of the 'Slopes' ('Type of Slopes'), similarly to the study of Robins, Haden and Garner (2006). What had been found, was that, in Semester-1 there was *neither a positive, nor a negative direction evident towards the learning of the topics of 'Arrays', 'Strings' and 'Collections'*. As Robins, Haden and Garner (2006, p.168) had designated, this neutrality in the 'Type of Slopes' most probably signified topics with inherently harder or unevenly distributed consequent labs, topics too difficult, or finally topics developing too quickly and in need of closer attention; designations which seemed to conform with similar literature findings, relating to novice programmers topics difficulties (i.e., Ehlert and Schulte 2009, p.24; Reges 2006, p.294; Robins, Rountree and Rountree 2003, p.162).

Lastly, while recapitulating, it is important to note here that it had been the results stemming from (Figure 3) graph's initial casual-inspection, and particularly, the phenomenon of Blackbox's data-findings portraying *'Arrays' having more Average-Number of Errors (per-user) than 'Strings' from Week-1 onwards* that had provided the stimulus for the readjustment of the project's focus; as well as, the drive towards the specific direction of the subsequent Chapters investigation. More specifically, after looking through a fair sample of Universities syllabuses[15], in which the instruction of 'Strings' was roughly placed between Weeks 2-3, while that of 'Arrays' between Weeks 6-7, this paradox had manifested; since someone, would logically expect to see inside the BlueJ IDE (used widely for first year beginner courses at universities)[16] – other Data-Structure topics being more error-prone during a Semester-1 curriculum's earliest weeks – like 'Strings', and *not 'Arrays'*.

[15] Al-Sammarraie (2016); COS 126: Precepts (Spring 2016) (2016); Nilsson (2016); O'Brien, (2016); Shkolnik (2016); Sullivan (2016); Chien and Haddad (2015); Drysdale (2015); Dupont, (2015); Lasseter (2015); McSweeney (2015); Lyman-Abramovitch, Higuera and Tremblay (2014); Wahls (2014).

[16] (Brown et al. 2014, p.223)

4. ArrayIndexOutOfBoundsException & 'Arrays' – Primacy in Difficulty

This Chapter conducts a cross-examination over certain pertinent smaller-scale literature studies, whether the primacy of the 'ArrayIndexOutOfBoundsException' class among the runtime exceptions, and the supremacy of the construct of 'Arrays' in respect to their difficulty over novice programming students' assimilation, are truly two accurate standpoints; a cross-examination that was deemed necessary, after taking into account the previous Chapter's findings, which have exhibited the ArrayIndexOutOfBoundsException as the class consisting the biggest percentage share of runtime errors (a percentage of 35%) – (Figure 1).

Thereafter, by beginning firstly with an examination over the analogous literature in the field of runtime exceptions, and particularly with the primacy of ArrayIndexOutOfBoundsException errors among runtime exceptions, despite the scarce publications − the phenomenon was generally apparent:

Pritchard (2015) compared and contrasted the most common error-messages between two different programming languages, Python and Java, in a creative primal attempt to examine the possible links between programming error-messages and statistical distributions. His Python error-messages were obtained from students' code-submissions in a website teaching introductory programming in Python named 'CS Circles', while for Java the error-messages were obtained from the student participants of BlueJ's Blackbox, as in this project. Hence, after examining his available online source-data[17], his runtime exception error classifications (Appendix F), were found to be completely identical with this dissertation's findings from Blackbox (Figure 1), placing the '*ArrayIndexOutOfBoundsException*' class in a *prime rank;* and validating thus in some informal way this study's MySQL retrieved data. Respectively, Thompson (2006) in her MSc Thesis, has also attempted to extract information about the runtime errors that novice programming students encounter; and she had done this, using the Mylar monitor plug-in for the Eclipse IDE and two additional monitors added over it (i.e., a runtime and a compilation error monitor), to extract information from students, as these were interacting with the Gild Eclipse IDE plug-in. Thompson (2006) in her Thesis has examined an only small group of subject-users, unlike Pritchard (2015) and this project that have used the Blackbox server[18]; but nonetheless, her findings have paradoxically complied with both studies, in exposing the '*ArrayIndexOutOfBoundsException*' class as a *leading* culprit in the hierarchy of the *runtime exceptions* (Table 2).

Notably, an interesting outcome that has resulted from her study, and that has proven to be compatible with both Pritchard's (2015) suggestions but also with this project's findings, was that 'a small number of error-types accounted for most of the occurrences' (Spohrer and Soloway 1986; Thompson 2006, p.92). Hence, looking at the runtime exceptions in Table 2 of Thompson's (2006) survey, it was observed that her first 2 runtime exception classes only

[17] `http://daveagp.github.io/errors/java.html`

[18] and which according to Jadud and Dorn (2015, p.131) could capture data of nearly 2 million programmers annually

(i.e., the NullPointerException-40% and the ArrayIndexOutOfBoundsException-27%), have accounted for a large 67% share of the overall exceptions made by all her participants; outcomes in perfect accord with this project's respective findings, both in terms of the largest instances of runtime exception classes, as well as in terms of the large percentage shares of these specific exception classes; i.e., in Figure 1 ArrayIndexOutOfBoundsException-35% and NullPointerException-34% only − accounting for a large 69% share of the overall runtime exceptions, made by all the Blackbox participants.

Count	%	Error type	Notes
184	40%	Null pointer	Assignment 1-4, Lab 5
124	27%	Array index out of bounds	Assignment 1-4, Lab 4, 7
117	26%	Unresolved compilation problem(s)	Usually decreases in occurance
8	2%	Class cast	Assignment 2, 4, 6
7	2%	String index out of bounds	Assignment 4, 5
7	2%	No such element	Assignment 4
4	1%	Stack overflow	Assignment 2, Lab 7
1	<1%	Out of memory error: Java heap space	Assignment 3
1	<1%	Regular expression: pattern syntax	Assignment 4
1	<1%	Number format: for input string	Assignment 6
1	<1%	Test: PASSED numeric java.lang.NullPointerException	Assignment 1, test case

Table 2 - Runtime exceptions for all participants (Thompson 2006, p.75)

Secondly, likewise while looking at the literature evaluating novice programming students most difficult topics, the predominance of 'Arrays' as the most challenging theme − was once more evident:

Ehlert and Schulte (2009) in their study have compared the 'Objects-First' vs. 'Objects-Later'[19] teaching paradigms[20], having as their ultimate goal to evaluate the impact that the two opposing approaches have over their novice programming students learning gain; hence, after having carefully designed a fair-comparison setting in their two approaches evaluation that would permit an easier interpretation of their results, they've discovered among others that irrespectively of the paradigm used, "*arrays*" and "associations" of classes (which are usually implemented using arrays) were the students' *most difficult topics* to learn (Ehlert and Schulte 2009, p.21).

[19] (sometimes also referred to as 'structured', 'procedural' or 'Imperative' approach)

[20] The differences between the two opposing approaches to the teaching of the CS1 course, pertain to the fact that there is no agreement among CS educators on the correct order in which to teach the introductory Java course's material; while some instructors propose the 'Imperative-First' paradigm, which suggests that the emphasis in the early topics should be placed on the imperative aspects of the language (i.e., expressions, control structures, procedures and functions) (AIJTF 2001, p.29), the proponents of the 'Objects-First' paradigm oppositely claim that the emphasis in the early topics should be placed on the object-oriented programming topics (i.e. classes and objects).

topic	max	points
T5: Control structure selection	10	7,0
T2: Data structures and control structure sequence	10	6,8
T1/T3: Introduction in the OOP / Class and object	20 (10)	13,2 (6,6)
T8: Inheritance	10	5,9
Addition: OOM	10	5,8
T6: Control structure loops	10	5,6
Addition: OOP (dynamic)	10	5,2
T4: Methods	10	5,0
T7: Arrays and Strings	10	3,7
T9: Association	10	2,1
Total	110	60,3

Table 3 - Ehlert and Schulte (2009, p.20) study's list of grade results from both their participant groups, ranked according to their difficulty (* easier topics at the top and more difficult topics at the bottom).

In continuation, Hyland and Clynch (2002) in their qualitative study, have tried to measure the effectiveness of various pedagogical initiatives employed in the teaching of Java at the Institute of Technology Tallaght's computing department; and additionally, to identify the concepts of Java that their first and second year college students find most difficult to understand. What they've discovered – was that '*Arrays*' were *the most difficult learning topic* of their survey's 90 participant-students (Table 4). Whilst finally, Meisalo, Sutinen and Torvinen (2003) at the University of Joensuu, have investigated (among others) the reasons, why their newly introduced university's distance-learning education program (addressed to Joensuu's surrounding rural-areas high-school students), had been causing high dropout rates, especially to their university's virtual CS1 Java programming-course pupils; an analysis, within the topics that constituted their virtual CS1 Java programming-course's pupils *biggest hurdles* (conducting them to this accentuated dropout phenomenon) – had revealed once more the prevalence of 'A*rrays*'.

Arrays	28%
Threads	20%
Polymorphism	14%
Looping Statements	10%
Exceptions	9%
Inheritance	6%
Objects and Classes	4%
Advanced Language Features	3%
Others combined	6%

Table 4 - Students 'most-difficult' learning topics (Hyland and Clynch 2002, p.104)

After having looked at the nature of the data structure of 'Arrays' from all the above case-studies, from the remarks made by several scholars[21], and as it will be further illustrated in

[21] (Hanks and Brandt 2009, pp.27-28; Ventura, Egert and Decker 2004; Kölling 1999a, p.13; Clark, MacNish and Royle 1998, p.174; Biddle and Tempero 1998, pp.49-52)

the following Chapter 5 (through a more meticulous inquiry into the array's constituent sub-constructs' categories responsible for generating most ArrayIndexOutOfBoundsException errors) − the topic of 'A*rrays*' could be admittedly characterised as *a particularly problematic theme for novice students*.

> The inclusion of some topics, such as arrays and loops, is surprising given that they are typically regarded as very difficult concepts in introductory programming (Robins, Rountree and Rountree 2003, cited in Robins, Haden and Garner 2006, p.170).

> '"arrays" do represent a real hurdle for the novice programmer'
> (Boulay 1986, p.67).

5. COMMONEST 'ARRAYS' CONSTRUCT'S ERRORS

This Chapter exhibits several further insights about the beginners misconceptions with the construct of 'arrays', and specifically relative to their interactions with the 'array's' most defective constituent sub-constructs' categories (i.e., those array related constructs' composing the 'array's' didactic material that conduct novices to generate the most ArrayIndexOutOfBoundsException errors). A final analysis on BlackBox's most defective 'runtime exception errors', which in this Chapter has been carried-out looking into the novices actual programming source-code (their 'actual `.java` source-file *contents*'); exploiting Blackbox's capability to capture the complete source-code text from those participants' errors whose Java packages' source-files upon loading in BlueJ contained the complete text content, and obtaining a fair sample (863) of ArrayIndexOutOfBoundsException`.java` files – to investigate the novices pitfalls' patterns with these alternative sub-constructs'.

Hence, in continuation, having begun with a brief overview upon the construct of 'arrays', the array unlike the set of collection types of the Java collections framework (i.e., Set, List, Map), was indicated as a *low-level* language construct of a static memory size (Dotnetperls.com 2014; Sternberg 2014); and was furthermore exhibited, as a very common non-primitive *data type* (Biddle and Tempero 1998, p.49), frequently appearing in the methods of the Java API (Ventura, Egert and Decker 2004, p.72), and customarily used for the implementation of Java programs cooperatively with the combination of other constructs. Something, for which Simon et al., (2010, p.216) pertinently to this latter fact, have specifically drawn the attention, suggesting that the difficulty in apprehension of constructs – was growing proportionately along with the concatenation of more combinations of constructs.

> In general, *a question that requires use of more than one construct might be considered more challenging* than one which requires only understanding and application of one. For example, a loop problem with an array may be more complex than a simple loop. The addition of an if-statement to that question may make it even more complex. ... *Arguably, having to employ more than one concept to solve a question does make that question more challenging* (Simon et al. 2010, p.216, my italics)

Question	Correct
If statement design (2Qs) (when to use if, else-if, else, compound if statements)	94%
Nested Loop iterator (2Qs) (trace)	92%
Class Design/language issue (5Qs) (select code)	85%
Find index of max element in array (1Q) (select code fragment)	74%
Loop over array replacing every other element with its index (1Q) (trace)	69%
Complex array reverse, with two iterators (1Q) (trace)	67%

Table 5 - High success values at the top designating students comprehension of single constructs, and low success values moving towards the bottom, designating weaker-comprehension of combinations of constructs (Simon et al. 2010, pp.216-217)

In consideration of all the above, at the onset of this uptake, an inspection towards the array's adjoining didactic material from CS1 courses syllabuses and Java introductory textbooks was deemed necessary, and had been undertaken[22]; in the interest of the identification of the array's constituent sub-constructs' categories, and of the application of an analogous classification to the source-file contents' error-cases, after their retrieval from Blackbox. As a result, this investigation amongst the array's material had generated the following subject categories (and parallelly potential misconception sources):

- TWO-DIMENSIONAL ARRAYS (a.k.a., MULTIDIMENSIONAL ARRAYS)

  As a container object of sequenced elements of a fixed-sized single *data type*, the array takes the form of either an one-dimensional (simplest form) or a two dimensional array (a.k.a., a multidimensional array – an array of arrays); and is comprised of a given '`array_name`' (whose elements are referred to by their indices), a '`type`' specifying the type of the elements the array will hold (i.e., primitive or reference type), and a '`size`' which decides the number of elements the array will hold (Saxena 2013, pp.245-247).

  ```
  type array_name[size];                        (1-D array)
  type array_name[row_size][column_size];       (2-D array)
  ```

  A two-dimensional array construct according to Sedgewick and Wayne (2016), is used in Java as a means to aid the implementation of those applications, which are seeking a natural way to organize their numerical information in a rectangular table format (similarly to a mathematical matrix); and is comprised of elements referred to by their rows and columns positions. For Example: The construct below, creates a two-dimensional array named 'arr' with two dimensions: a first to specify that 'arr' has 2 rows of elements, and a second to specify that each of those elements are themselves an array of 3 columns of elements:

  ```
  int arr[2][3];
  ```

  Nonetheless, despite the multi-dimensional arrays encomiums in terms of their: efficiency in modelling data structures of multiple dimensions at a higher level of organisation, enhanced navigation, ease of maintenance and increased performance (Collins 2003), multi-dimensional arrays were concurrently indicated as: complex (Dotnetperls.com 2014); "error-prone" (Siegfried, Chays and Herbert 2008, p.20); harder for beginners to understand (Boulay 1986, p.67), and frequently instructed by problematic and non-stimulating exercises, like the *grade book* and the *Matrix* class programs (Ventura, Egert and Decker 2004, p.70; Burger 2003, p.205). Instructed, with non-challenging exercises instead of more fun ones, like those suggested by Burger (2003) of *graphical 2d-array image manipulation* assignments, which had conversely proved to increase the students interest and to improve their CS1 pass-rates (Guzdial 2015, pp.57-63; Simon et al. 2010, pp.216-217).

- ARRAYS OF OBJECTS

  As was already stated above, the '`type`' of the array will specify the type of the elements that the array will hold (i.e., primitive or reference type); and while above

[22] (Wallace [2016]; Williams [2016]; Stepp and Martin [2015]; Deitel, Deitel and Deitel [2012]; AMBEDKAR [2011]; Slee [2006])

an example was shown of a 2D-array declaration for which all elements of the array stored had been specified to be of the integer (`int`) primitive type: `int arr[2][3]`, as with primitive types (int, String, double, etc.), an array is also capable of storing values where each element of the array is based on a formal class (i.e., '`type`' being an object). For Example: Presupposing the existence of a class 'Student', in the example below, just like for an array of primitive type data, an array is created which can hold references to seven 'Student' objects - an array of objects:

```
                                type      array_name          [size];
class Student {
   int marks;        ——→   Student[] studentArray = new Student[7];
}
```

However, the construct of 'arrays of objects' appears to introduce its own difficulties[23], by perplexing students about how to initialize its members (Hazzan, Lapidot and Ragonis 2011, p.185); and by expecting them to have some prior-knowledge of object-oriented programming constructs[24] and of advanced design techniques like composition (e.g., accessing an element of an array and an instance variable of an object: studentArray[2].marks;).

- COMMAND-LINE ARGUMENTS

  According to usadye.ru (2015), command-line arguments are simply defined as an alternative way of specifying configuration properties for a Java application; rather than clicking the application's icon from the operating system, the application is run instead from a terminal window, with the user typing the application's name followed by several necessary values (for the application to run), which are then passed to the application's starting point (i.e., the main method). These values typed in the terminal window (exempt from the application's name), are called *command-line arguments* and are being stored inside the main method's String array (`args[]`), with the first value (that is, after the name of the program) being stored at index 0, the second at index 1, etc. Based on this, inside the main method's executable set of statements, the first argument is represented by `args[0]`, the second by `args[1]`, etc.

```
public static void main(String args[]) {
BufferedReader in  = new BufferedReader(new Input …(System.in));
String lastName    = args[0];
double Hours = Double.parseDouble(args[1]); // set of statements
…
}
```

However, due to novice students inexperience with arrays, either by failing to configure appropriately inside their main method's[25] executable set of statements their `args[]` elements, or alternatively, by failing to insert in their terminal window

[23] (Robins, Haden and Garner 2006, pp.167-168)

[24] to be able to eschew problems like the 'parallel arrays of information' designated by Horstmann (2009, p.280) as a frequent error-case.

[25] The very first method that novice students are introduced to, and which per se is already convoluted with advanced concepts (i.e., encapsulation, variables scope, parameter-passing and arrays), making the novice students feel intimidated (Kölling 1999a, p.13; Clark, MacNish and Royle 1998, p.174; Biddle and Tempero 1998, pp.51-52).

the right number of values that the Java application is expecting to run correctly[26] – ArrayIndexOutOfBoundsException errors ensue.

- PASSING/RETURNING ARRAYS IN METHODS

  Although the main purpose of an 'array' is to provide access to various values as a container object which efficiently stores and manages data grouped under one name, the array is primarily a variable; and as such, it can be still both *passed to* a method and *returned from* a method, similarly like with any other regular Java variable.

```
public void showPoints(int[] Points){}  // Passing an Array To a Method
public int[] Initialize(){}           // Returning an Array From a Method
```

  Of particular interest is the fact that exposure of 'arrays' to other classes (tight association of arrays with methods operations), has been acknowledged as an unsafe (Sternberg 2014) and a hindering towards students understanding practice (Bruce, Danyluk and Murtagh 2005); but that nonetheless still, as a part of this investigation into the array's adjoining didactic material, the pertinent to the 'methods' construct's topic of: 'Passing, Returning Arrays To/From Methods', was emphatically found to constitute a typical instruction subject.

- FENCEPOST PROBLEM

  Although Java's creators have designed the language's run-time system settings to ensure programmers indices are within the bounds of the array (bounds checking mechanism), nonetheless they have missed the opportunity to provide a 'fix' to the language's proverbial indexing mechanism (i.e., array's subscript indexes still begin from zero and end with the array's size minus one) (Lewandowski n.d., p.14); a trait which was indicated to extremely confuse novice students working with arrays (Farrell 2013, p.160; Scott, Watkins and Mcphee 2007, p.4; Tew, McCracken and Guzdial 2005, p.31). Thus, most prominently in cases of initializations of arrays through the definite `for` loop construct, the fallacious interpretation of how many times a loop should iterate (i.e., causing a loop to execute one time more or less than intended), would lead to an ArrayIndexOutOfBoundsException logic error known as a: *Fencepost* or *Off-By-One* Error. For Example: Considering that arrays subscript indexes start from 0 instead of 1, the mistaken use of the "<=" instead of the "<" relational operator inside the array's initialization in the `for` loop below, would eventually conduct in an ArrayIndeOutOfBoundsException Fencepost error; to indicate that the result for `array[5]` cannot be displayed, since the array's upper bound is only 4.

```
int array[] = new int[5];

for (int i = 0; i <= 5; i++)
   System.out.println(array[i]);
```

- ARRAY INITIALIZATION

  The examination of the arrays adjoining didactic material had revealed, that a very basic sub-construct (for which there was consistently a brief reference), was that relevantly with the concept of 'arrays initialization'. A reason which could probably

[26] (i.e., typing less or more command-line arguments than those that are necessary.)

justify the subject's omnipresence/emphasis among the didactic material, could potentially be due to the fact, that in several Java introductory textbooks (Horstmann 2012, pp.250-256; 2009, p.280; Eckel 2006, pp.66-67), 'arrays initialization' is exhibited as a very common programmers error; and more specifically, the error that programmers make is trying to assign to an array's element a value without having first initialized the array (i.e., without having primarily instantiated the array with the `new` keyword – which creates the actual array object and allocates memory to its elements).

```
double[] values;
values[0] = 29.95;                // Error! values uninitialized
double[] values = new double[10]; // values initialized
```

- `java.util.Arrays`

  The special utility '`java.util.Arrays`' class of the `java.util` package (a topic which periodically appears among the 'arrays' CS1 course's adjoining didactic material) is a set of static 'ready-made' methods inside the Java SE's standard library, offered as a support towards Java's programmers to help them perform certain common array manipulations; e.g., copy, fill-in, sort, search, and compare values in arrays (without them having to 'reinvent the wheel'). For programmers to be able to use these methods, they must initially import the `java.util.Arrays` class in their classes.

- ENHANCED FOR STATEMENT

  As a typical unit of array's didactic material, the *enhanced for-loop* (a.k.a., *for-each* loop) had been introduced in JDK 1,5 as a new (`for`) loop construct to simplify the processing of arrays; by offering an easier way of iteration through an array's elements, without requiring the use of the traditional `for` loop's counter, and neither of the array's indexes. However, the fact that the new `for` loop construct had depleted from its syntax the use of a counter, though from the one hand has aided in eliminating the presence of arrays 'out-of-bounds' errors, from the other hand it had confined its use only for cases where an array could loop through and have its element values read (e.g., sum its integer values as in the code below); in cases where an array would need to loop through its element values, and have them modified, the normal `for` statement would still be necessary (Deitel, Deitel and Deitel 2012, pp.258-259). The appropriate use of a loop in scanning all the array's elements (e.g., in search algorithms), had been indicated as a common students' misconception (Hazzan, Lapidot and Ragonis 2011, pp.185; Smolarski 2003, pp.142-143).

```
int[] numbers = {8, 2, 6, 4, 3};
int sum = 0;
for(int number : numbers){ // for each int number in int[] numbers
   sum += number;
}
```

- MISCALCULATED BOUND

  Lastly, a general category termed 'Miscalculated Bound' had been comprised to encompass all the array's complementary didactic material concepts, like those indicated by Hazzan, Lapidot and Ragonis (2011, p.185) (e.g., basic scans, tasks that implement different algorithmic approaches of different logic complexity, tasks that involve building, search, sort, merge, etc.); and although this category's error-cases all revolved around a central theme: 'misuse of an array's index value', its existence was deemed necessary, considering those complementary materials ramifications on students miscomprehensions, as exhibited by various scholars like Hazzan, Lapidot and Ragonis (2011, pp.185-186), whose inquiries had revealed numerous additional frequent students' mistakes upon arrays; e.g., confuse an array element's subscript with its value[27], exceeding an array's index beyond the array's size, overwrite an array element's value while sorting, leave empty elements instead of writing values successively when building a new array – like in an intersection, etc..

Thus, having completed the inspection of the array's constituent sub-constructs' categories, following a visual inspection of the ArrayIndeOutOfBoundsException class's 'Average-Number of Errors (per-user)' trend in Figure 3, and observing that the trend had an ascending pattern for weeks 1-7, it was decided to further examine Blackbox's source-file contents' error-cases for this sub-period's initial (Week-1) and last (Week-7) weeks independently. This analysis, which would compare the Relative Frequencies of the alternative constituent sub-constructs' categories of 'array' errors, separately for Week-1 and for Week-7 (as well as of their total Rate of Change between weeks 1-7), would empower a rationalization upon *the primary reasons responsible for the phenomenon of the constantly increasing number of array errors from Week-1 to Week-7*; an analysis, which was eventually performed, displaying/comparing the percentages of users whose exception errors had emerged for using:

(1) BlueJ in an inconsistent (non 'objects-first') manner by not eschewing Java's '`main{}`' method (Brown et al. 2014, p.225), 'array algorithms', and 'array algorithms using recursive methods' (Table 6 & Figure 7).

(2) Any of the array's aforementioned sub-constructs' categories (Table 7 & Figure 8).

(3) The multidimensional array sub-construct's typically instructed exercises (Table 8).

[27] also supported by (Scott, Watkins and Mcphee 2007, p.4; Boulay 1986, p.66)

	Week 1	Week 7	Rate of Change
Proportion of 'main{} method' cases	78.1%	83.0%	4.9%
Proportion of 'algorithm' cases	35.2%	49.0%	13.8%
Proportion of 'recursive algorithm' cases	4.9%	6.1%	1.2%

Table 6 – Relative Frequencies (percentages) and Rate of Changes of the complete structure of 'arrays' sample error-cases (* to represent the changes of the structure of 'arrays' frequencies in a more visually perceptible manner, Figure 7 had been complementarily plotted).

	Arrays (type) frequency - Week 1	Arrays (type) frequency % - Week 1	Arrays (type) frequency - Week 7	Arrays (type) frequency % - Week 7	Rate of Change of sub-constructs errors from Week 1 to Week 7
`Arrays of Objects`	43	6.98%	18	7.29%	7.79%
`command args[]`	97	15.75%	11	4.45%	-70.80%
`Enhanced for Statement`	4	0.65%	2	0.81%	28.74%
`FENCEPOST`	106	17.21%	30	12.15%	-27.13%
`Java.util.Arrays Class`	4	0.65%	1	0.40%	-35.63%
`Miscalculated Bound`	159	25.81%	60	24.29%	-2.83%
`Multidimensional Arrays`	110	17.86%	55	22.27%	28.74%
`Passing/Returning Arrays in Methods`	84	13.64%	70	28.34%	114.57%
`uninitialized variable`	9	1.46%	0	0.00%	-100.00%
Total	616	100.00%	247	100.00%	

Table 7 – Relative Frequencies (percentages) and Rate of Changes of the alternative constituent sub-constructs' categories of 'arrays' (* to represent the changes of the categories frequencies in a more visually perceptible manner, Figure 8 had been complementarily plotted).

	Multidimensional Arrays (type) frequency - Week 1	Multidimensional Arrays (type) frequency % - Week 1	Multidimensional Arrays (type) frequency - Week 7	Multidimensional Arrays (type) frequency % - Week 7	Rate of Change of multidimensional arrays errors from Week 1 to Week 7
GradeBook	2	4.44%	1	4.76%	7.14%
Java Image I/O API	4	8.89%	8	38.10%	328.57%
Matrix	39	86.67%	12	57.14%	-34.07%
Total	45	100.00%	21	100.00%	

Table 8 – Relative Frequencies (percentages) and Rate of Changes of the alternative typical exercise themes' categories of 'multidimensional-arrays'.

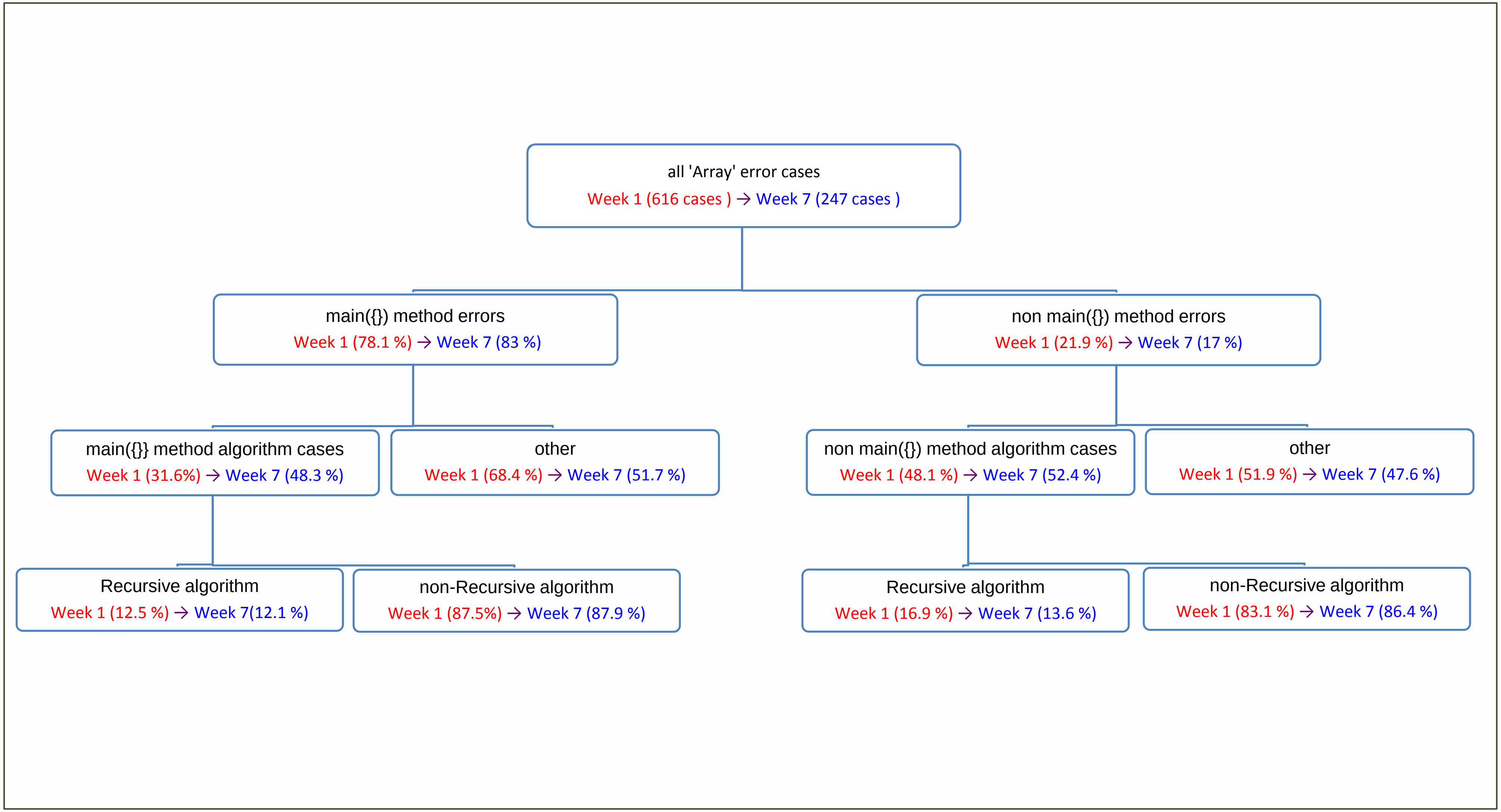


Figure 7 – Complete structure of 'arrays' sample error-cases for Week 1 and Week 7.

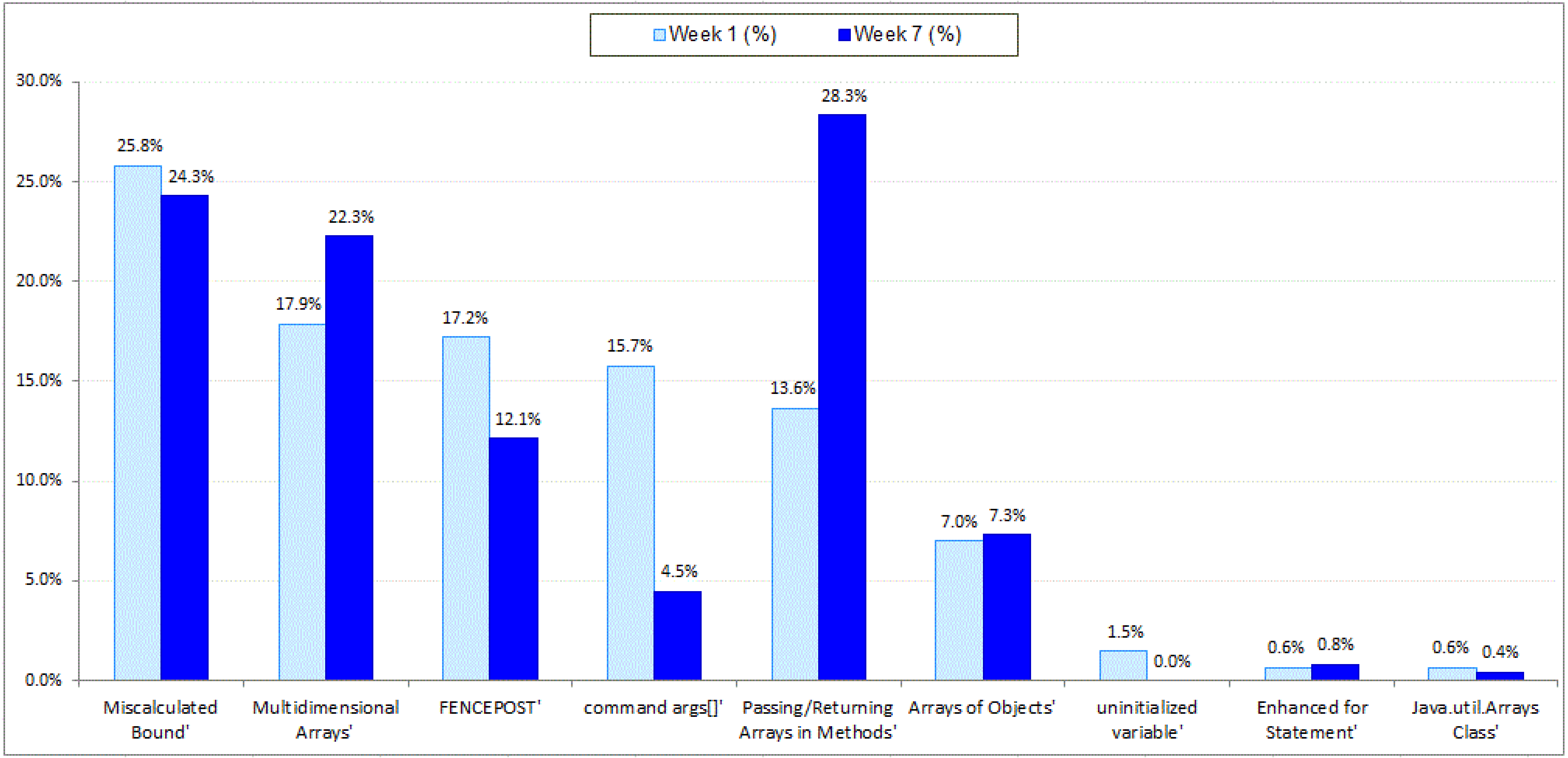


Figure 8 - Relative Frequencies (percentages) of the constituent sub-constructs categories of 'arrays' for Week 1 and Week 7.

Synoptically the Tables 6-8 & Figures 7-8 above, have exhibited the following significant outcomes:

A. Despite *the BlueJ IDE* being recommended as a development-framework to facilitate the instruction of Java through the 'Objects-First' teaching approach (Kölling et al. 2003, p.252), the Week-1 and Week-7 sample Java source-file error-cases (863) examined in Table 6 & Figure 7, have revealed that *a large (approx. 81%) share of students had been using it in a non-indicated way* - not eschewing Java's `main`{()} method. More particularly, the sample error-cases have shown that, in the critical first weeks of the CS1 course's Semester-1[28], only a small number of students were being instructed using BlueJ appropriately via an 'Objects-First' approach (focusing on BlueJ's UML graphical representations); and that oppositely the majority was being taught through a 'Structured Programming' style, placing all their code in a single class's '`main`{()}' method, and implementing programs focusing on BlueJ's text-based editor (Pasa Uysal 2012, p.818; Pont 1998, p.22), *requiring them a steeper learning curve* (Kölling 1999b, p.7). Interestingly, the large identified percentage (81%) of 'structures-first' paradigm like data-findings' inside Blackbox's BlueJ Java source-files' errors, was also found to comply with the remarks made by Kölling (2015, pp.13-14), as well as those of Bruce (2004, p.33)[29] – who (the latter) in his paper upon pondering over the issues raised in the way of teaching introductory courses with Java, had pointed towards something alarming in regards to the origin of errors:

> One contributor to the discussion pointed out an Australian survey that showed *a mismatch between the languages taught and the programming paradigm taught*. Over 80% of classes used an object-oriented language, but more than half of the faculty chose to teach using the procedural paradigm (Bruce 2004, p.33, my italics).

B. The investigation among the 'arrays' adjoining didactic material had shown (Table 7) that the array's constituent sub-construct category of: 'Passing/Returning Arrays in Methods' was the most problematic concept in students learning (exhibiting the highest rate of change from Week-1 to Week-7 – 114.57% more array errors); students for programs making use of *arrays' parameter-passing/returning in methods, had more than doubled their array errors* (Figure 8). An outcome which however, did not come as a surprise, considering the practice of 'parameter-passing/returning of arrays in methods' negative effect on novices as exhibited by Bruce, Danyluk and Murtagh (2005)[30], as well as its highly erroneous nature as a construct itself as indicated from the studies of (Tew 2010, pp.56-59; Dale 2006, p.51; Thompson 2006, p.91; Sayers, Nicell and Hagan 2003, pp.107-108).

C. In Table 8, the examination among the alternative exercise themes' categories pertinent to multidimensional arrays (i.e., Gradebook, matrix, and Image I/O API

[28] (Robins 2010, p.64)

[29] Further particularly evident in the study of Donchev and Todorova (2008, p.169).

[30] In the study of Bruce, Danyluk and Murtagh (2005, pp.246-249), novice students had demonstrated a tendency to write array programs with a less superior organization, by not encapsulating them in classes, but instead by making them available globally as instance variables, and passing them around as parameters in methods (methods tightly associated with array operations).

assignments), has revealed a gradual decrease (-34.07%) in the number of errors relevant to matrix exercises[31], and a simultaneous gradual *increase in the number of errors relevant to graphical 2D-array image manipulation exercises*. Essentially, an indication to demonstrate primarily the gradual evolvement of a higher number of students towards the 'Media Computation' contextualized teaching approach[32]; and consequently also, to partially justify the stalling of the further upraise of array errors after Week-7 in Figure 3 (considering the 'Media Computation' paradigm's lack of 2D array-indexing syntax material (Simon et. al. 2010, p.217) – which as seen from the exception-messages in Figure 2 constituted the biggest students hurdle).

PA\Week		1	2	3	4	M	6	7	8	9
If statements	T									
	MC									
Loops	T									
	MC									
1-D Arrays	T									
	MC									
Nested loops	T									
	MC									
2-D Arrays	T									
	MC									
Object Use	T									
	MC									
Class Design	T									
	MC									

Table 9 - Comparison of constructs in students assignments, depicting the span of 2D-arrays, between a Traditional objects-first (T) vs. a Media-Computation (MC) teaching approach in a CS1 curriculum (Simon et. al. 2010, p.215).

.

[31] (e.g., matrix-multiplication algorithm exercises typically emphasized in the initial weeks of CS1 courses following the 'Imperative-First' approach)

[32] Typically instructed through the introductory textbook of Guzdial and Ericson (2007), as cited by Simon, et al., (2010, p.214).

6. CONCLUSIONS & FURTHER RESEARCH

In this Chapter an overall appraisal over the Chapters' most significant outcomes will be exhibited; before concluding with some proposals for those researchers interested in delving deeper into Java's Runtime-Errors, and 'deciphering' further some of the behavioural novice students characteristics, relatively to their misconceptions with the programming constructs' identified to conduct them to most runtime exception pitfalls.

Hence, at this point in retrospect, looking-back over the materials inquired in their entirety, it could be unequivocally proclaimed that Blackbox's implementation team had been right to acquiesce that the Blackbox project is not a 'panacea' for programming research, and to recount its potential shortcomings[33]. Since, during the course of this project, the outputs have at times appeared to provide *vague results*; as upon the (eventually aborted) themes of:

- ☒ "*Prior programming experience's effect upon students' performance*". A considerable investigation had been performed over the hypothesis that the 3 exception classes 'neutral' 'Type of Slopes' in Table 1 (lack of significant change in Semester-1 towards a positive/negative learning direction), was actually reflecting the merged behavior of users with various prior-programming experience backgrounds, moving through the CS programming courses sequence, and having a performance convergence over their understanding of introductory topics (Tew, McCracken and Guzdial 2005; Holden and Weeden 2004). An assumption (theory), which unfortunately principally due to *Blackbox data's scarce information/anonymity of its participant-subjects*, had proven infeasible to validate, and had eventually led to this theme inquiry's termination.
- ☒ "*The introductory programming CS1 teaching approach* ('Objects-First' vs. 'Objects-Later') *exerting the highest learning gain to novice students*". Despite the highly suggestive indications deduced from a plethora of Universities syllabuses and Java introductory textbooks, that for the 'Objects-Later' approach 'arrays' are introduced between (Weeks 1-6)[34] while for the 'Objects-First' between (Weeks 7-onwards), this theme's substantial inquiry was brought to a halt; and specifically, due to the *lack-of-infos/anonymity of Blackbox's participant-subjects*', in addition to the discovery of *a number of peculiar Universities CS1 syllabuses curricula*[35] that had been found to follow obscure patterns in the order of 'arrays' instruction in their curricular topics sequences; inconsistent with most typical CS1 syllabuses[36] – jeopardizing thus, the subsequent assumptions objectiveness.

---

[33] (Jadud and Dorn 2015, p.132; Brown et al. 2014, p.225; Brown 2013)

[34] Further evidenced in Table 8, from the large number of multidimensional-array matrix algorithm exercises (a typical 'Objects-Later' approach's material), in the initial week of Semester-1.

[35] (OBJECTS-LATER/IMPERATIVE: Sprague and Chao 2016; Braffitt 2015; OBJECTS-FIRST/OBJECT-ORIENTED: Garbinato 2015)

[36] (OBJECTS-LATER/IMPERATIVE: Sheller 2015; Markov 2010, OBJECTS-FIRST/OBJECT-ORIENTED: Al-Sammarraie 2016; Latham 2015)

However, on the other hand, it is also similarly and utmost importantly true, that Blackbox's limitations were found to constitute an only minor hindrance, as compared to the usefulness that the provision of its pragmatic values (i.e., context of exception classes, messages, contents and Tables/Figures) have attained; and thus, several cases of:

- *'Experts blind spots':* discrepancies between what instructors think and what actually are the commonest novice-students' errors (Guzdial 2015, pp.32-33; Thompson 2006, p.3), *have been dissolved*; as for instance in Table 7, pertinent to:
  - ☑ The accentuated exhortations over the inappropriate use of the 'arrays' *enhanced for-loop* constituent sub-construct; which contrary to the anecdotal evidence of (Hazzan, Lapidot and Ragonis 2011, pp.185; Smolarski 2003, pp.142-143), juxtaposed against Blackbox's massive dataset of erroneous source-files' contents, had been identified to constitute a *non-regular* students' misconception field.
  - ☑ The extremely-accentuated exhortations over the inappropriate use of the '*arrays initialization*' sub-construct; which contrary to Eckel (2006, pp.66-67) and the textbook series of Horstmann (e.g., 2012, pp.250-256; 2009, p.280), juxtaposed against Blackbox's massive dataset of erroneous source-files' contents, had been identified (as with the *enhanced for-loop* case above), to constitute a *non-common* learners error field.
- *Previous literature case studies* conclusions (investigating those same Java constructs' types, albeit collected from *much smaller dataset samples or* from *empirical observations*), have been *validated*; and their small-scale formerly hypothetically biased results, had been reliably ratified as not such – rebutting thus, any doubts against the appropriateness of those findings more general (broader) applicability. As in:
  - ☑ Thompson (2006), who similarly to Spohrer and Soloway (1989), in her inquiry upon novice programmers errors, using a small dataset sample of 10 participant subjects, had asserted that "*a small number of error-types accounted for most of the novice-students mistakes*" (Table 2); demonstrating among others, how 2 runtime exception classes only (i.e., the NullPointerException and the ArrayIndexOutOfBoundsException), had been accountable for most of her participants' runtime exception errors. Outcomes, which were found to *comply perfectly with this project's respective findings*, as exhibited in Figure 1 – derived nonetheless, from Blackbox's huge dataset of thousands of participant-subjects.
  - ☑ (Ehlert and Schulte 2009, p.20; Howe, Thornton and Weide 2004, p.293; Ventura, Egert and Decker 2004, p.72), whose findings, deriving from empirical observations and from a small sample of university CS1 and secondary-school classes' participant subjects' datasets, had all pinpointed "*the problematic nature of 'arrays' when introduced quite early in the curricular topics sequence*". Findings, *conforming fully with this study's* 'ArrayIndexOutOfBoundsException' trend's constantly increasing 'Average-Number of Errors' *values* in the initial weeks (Week-1 to Week-7) of Semester-1; as exhibited in Figure 3 – stemming this time from the much larger dataset of Blackbox's thousands of participant-subjects.
  - ☑ (Kölling 2015, pp.13-14; Donchev and Todorova 2008, p.169; Bruce 2004, p.33), whose remarks based on certain anecdotal evidence, had adduced as a potential impediment towards the proper instruction of object-oriented programming, the apparition of a peculiar phenomenon of a large number of institutions' faculty, who were found to choose for teaching an object-oriented language but using a structured paradigm; "*large number of classes' having a mismatch between the languages taught and the programming paradigm taught*". Remarks, which were found to *strongly adhere* to this project's findings in Table 6 & Figure 7, which have exhibited how a large (approx. 81%) share of students, had been taught Java in BlueJ using a non-indicated structured style (by not

eschewing Java's `main{()}` method from their sources' code); yet, this time based on pragmatic evidence – retrieved from Blackbox's massive dataset of erroneous source-files' contents.

☑ Bruce, Danyluk and Murtagh (2005), whose findings based on a small sample of a CS1 course's students, have pinpointed "*the particularly negative effect that the practice of 'parameter passing/returning of arrays in methods' exerted on novices*". Findings, which were found to be in *perfect accord* with this study's results; as the data of the much more robust dataset of Blackbox's erroneous source-files' contents in Table 7 & Figure 8, have similarly identified the array's constituent sub-construct category of: 'Passing/Returning Arrays in Methods', as the most problematic concept towards students learning pertinent to arrays; (i.e., exhibiting the highest rate of change from Week-1 to Week-7 – 114.57% more array errors).

Finally, at this point it is important to note that amid this study's quest in comprehending more about the role of Java constructs as applied to students learning, it appeared that the 'Experts blind spots' dissolution, and the previous smaller-scale studies results validation/ratification for further generalization, *had indeed enabled the deduction of certain intriguing qualitative research insights*; exactly as Utting et al., (2012, p.4) had foreseen, when they had remarked about Blackbox's large scale multi-institutional/multi-national users data-set comparisons prospects. Insights, which nonetheless in this paper, had barely scratched the surface of the potential that the further elaboration of the voluminous data collected could provide, towards clarifying the beginner-learners 'grey areas' – leaving unanswered a number of significant questions.

Questions, indicatively, like those pertinent to:

⇒ The origin of difficulties for the second most erroneous runtime exception class, the NullPointerException class (Figure 1), which according to (Java With Us 2013; Wittman, Mathur and Korb 2013, p.181; Ben-Ari 2007, pp.18-19), could very likely be caused from the users omission in appropriately initializing arrays having elements of reference type (i.e., arrays of objects or `String` arrays); often, via the enhanced `for`-loop. A denotation, essentially indicating the exigency for *an additional retrieval/inquiry (similar to that of Chapter 5), this time, into the source-file contents of the NullPointerException errors*; towards an additional retrieval/inquiry, whose forthcoming prospective supplementary findings, could potentially annul/reformulate the contradictory 'Experts blind spots' conclusions that had emerged above – relatively to the excessively accentuated negative exhortations of the 'arrays initialization' and 'enhanced `for`-loop' array's constituent sub-constructs' categories.

⇒ The construct of 'arrays' targeted framework's residual context – *the more meticulous and exhaustive assessment of the culpable factors responsible for the novices misconceptions with arrays;* i.e., the classification one scale further, of the source-file contents sample error-cases of Chapter 5, among those '*definitely construct based*' '*deep*' (and harder for educators to remediate)[37] novices array

[37] Like those stemming from the mistaken belief that computers could reason with constructs' semantics similarly to how humans do, and which reflect novices' problems with the understanding of *the language's more fundamental constructs* (e.g., reading, branching, looping, etc.). On this occasion an interesting case-scenario would be, to examine the fifth Chapter's '*Common Array Algorithm*' constructs' source-file instances (Figure 7), and consequently, to validate the credibility of the assertion that the bubble-array sorting algorithm is counter-intuitive (less closer to how people intuitively sort items in real life), and harder to understand/implement when introduced as the very first sorting algorithm example in CS1 courses (Nieminen 2008; 2006).

errors, and those simpler '*surface*' ones[38] (Putnam et al. 1986, pp.460-461; Stanford University School of Education 1984, pp.27-28). A meaningful prospective reappraisal idea, which had stemmed from the assertions of Spohrer and Soloway (1986, pp.629-631) – that some factors causing novices to generate errors initially appearing as '*definitely construct based*', could actually be attributed instead upon the novices misunderstandings of conceptual model representations (Plans) ('*plan composition problems*').

⇒ *The identification of the remaining two most problematic novices basic Java constructs (i.e., 'Strings' and 'Collections') most defective constituent sub-constructs' categories*; since their fairly anticipated augmented number of errors for the upcoming year (illustrated in Figure 9 & Figure 10), had deemed the exigency for an inquiry similar to that of Chapter 5 imperative. The assumption that in year (2016) an increase in the number of the two classes erroneous data is expected to ensue, was based on the fact that in Figure 9 & Figure 10, following a Forecasting Time Series analysis[39], the actual and the predicted (Forecast) monthly instances were found to be very proximate (predictions had been very accurate); something, which had consequently supported the hypothesis that the same trend of years (2013-2015) was most likely going to continue for the following year (2016) – as with the predictions made by Jadud and Dorn (2015, p.138).

In any case, during the course of this project, BlueJ's Blackbox project (irrespectively of its few limitations) has proven overall, capable of providing enlightening quantitative and qualitative research insights, and to affirm its supportive role towards the synthesis of this study's framework of the commonest Java constructs pitfalls. It is certainly a promising project, having demonstrated its *efficiency in shedding light upon a vast spectrum of 'grey areas' about the way beginners interact with Java*, similarly to its use in previous studies (Altadmri and Brown 2015; Pritchard 2015; Brown and Altadmri 2014); and one whose future undertakings are earnestly anticipated within BlueJ's wide users range.

[38] Like those ensuing from the novices' faulty knowledge of the language's syntactical features (e.g., FENCEPOST array's sub-construct category, assignment statements, etc.).

[39] (Jalayer Academy 2013a; 2013b)

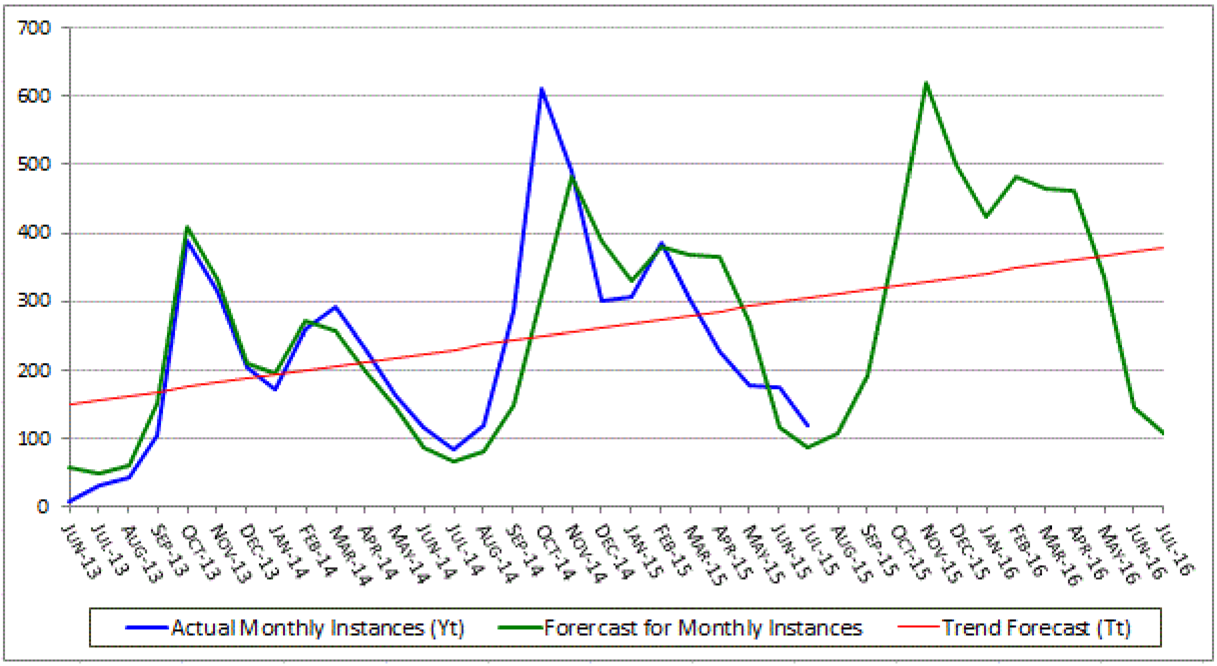


Figure 9 – Prediction of StringIndexOutOfBoundsException class's Average-Number of Monthly Error instances for 1 extra year

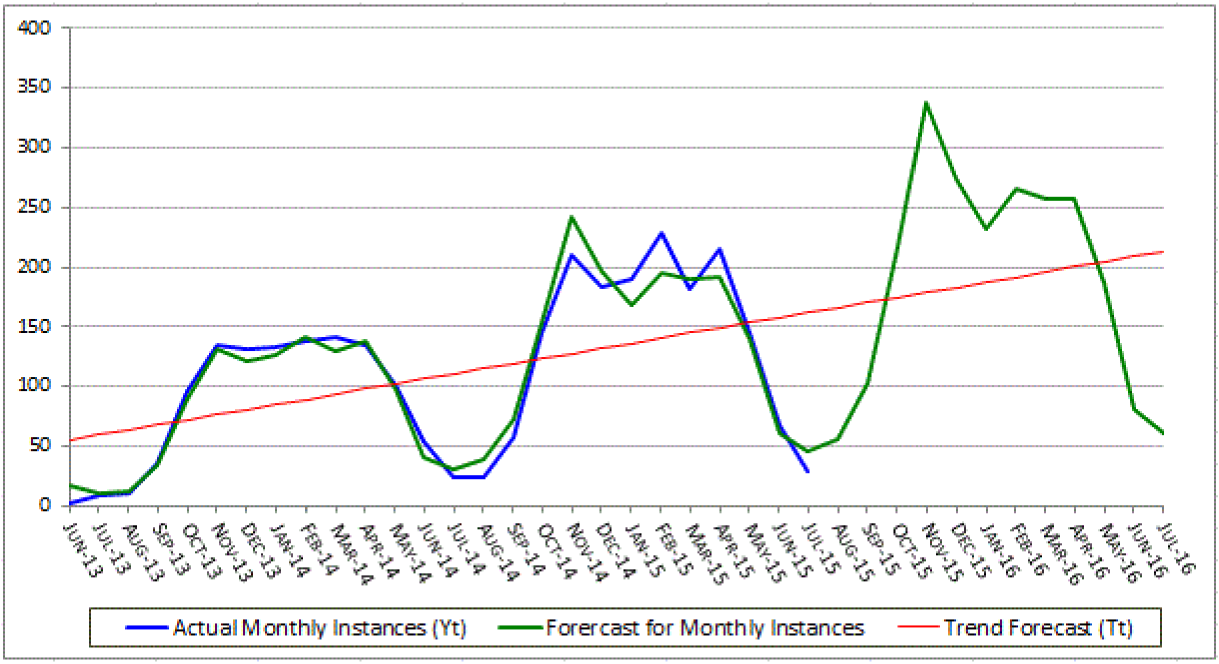


Figure 10 – Prediction of IndexOutOfBoundsException class's Average-Number of Monthly Error instances for 1 extra year

REFERENCES:

ACM/IEEE-CS Joint Task Force on Computing Curricula (AIJTF), (2001). *Computing Curricula 2001. Computer Science. — Final Report —*. [Online] Available from: http://www.acm.org/education/curric_vols/cc2001.pdf [Accessed 1 Jan. 2016].

Akraju, G. (2008). *logic problem or thread problem? (Swing / AWT / SWT forum at Coderanch)*. [Online] Coderanch.com. Available from: http://www.coderanch.com/t/345964/GUI/java/logic-thread [Accessed 29 Aug. 2016].

Alexander, A. (2010). *Android example source code file (PackedIntVector.java)*. [Blog] alvinalexander.com. Available from: http://alvinalexander.com/java/jwarehouse/android/core/java/android/text/PackedIntVector.java.shtml [Accessed 23 Sep. 2015].

Al-Sammarraie, R. (2016). BCIT : : Course Outlines. [Online] Bcit.ca. Available from: http://www.bcit.ca/study/outlines/20162062079 [Accessed 2 May 2016].

Altadmri, A. and Brown, N. C. C. (2015). 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. In: *SIGCSE '15 The 46th ACM Technical Symposium on Computer Science Education*, 04 - 07 March 2015, Kansas City, MO, USA. pp. 522-527

AMBEDKAR, B. (2011). *DR. BABASAHEB AMBEDKAR MARATHWADA UNIVERSITY, AURANGABAD DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY SCHEME FOR CHOICE BASED CREDIT SYSTEM (CBCS)*. 1st edn. [ebook] Aurangabad, Maharashtra, Republic of India: MARATHWADA UNIVERSITY, pp.52-53. Available from: http://bamu.ac.in/dept/csit/Upload/msc_cs_it.pdf [Accessed 24 Mar. 2016].

amin, m. (2014). *error string index out of range*. [Online] Javaprogrammingforums.com. Available from: http://www.javaprogrammingforums.com/whats-wrong-my-code/38348-error-string-index-out-range.html [Accessed 9 Sep. 2015].

Annotated SPSS Output: Regression. (2014). [Online] UCLA: Statistical Consulting Group. Available from: http://www.ats.ucla.edu/stat/spss/output/reg_spss.htm [Accessed 31 Jan. 2016].

Barland, I. (2008). Some Methods for Teaching Functions First Using Java. In: Hendrix, D. and Lester, C. ed. *ACM SE '08 ACM Southeast Regional Conference*, *Auburn, CA, USA March 28-29, 2008*. NY, USA: ACM New York, pp.256-259.

Batool, A., Rehman, M., Khan, A. and Azeem, A. (2015). Impact and Comparison of Programming Constructs on JAVA and C# Source Code Readability. *International Journal of Software Engineering and Its Applications*, 9(11), pp.79-90.

Beginnersbook.com, (2014). *Checked and unchecked exceptions in java with examples*. [Online] Available from: http://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/ [Accessed 7 Dec. 2015].

Ben-Ari, M.M. (2011). *Compile and Runtime Errors in Java*. Houston, Texas: Rice University. Available from: https://archive.org/details/ost-computer-science-col10913 [Accessed 28 March 2015].

Ben-Ari, M.M. (2007). *Compile and Runtime Errors in Java*. [online lecture] 24 January 2007. Rehovot, Israel: Weizmann Institute of Science. Available from: http://introcs.cs.princeton.edu/java/11cheatsheet/errors.pdf [Accessed 14 March 2015].

Biddle, R. and Tempero, E. (1998). Java pitfalls for beginners. *SIGCSE Bull.*, 30(2), pp.48-52.

Böszörményi, L. (1998). Why Java is not my favorite first-course language. *Software - Concepts & Tools*, 19(3), pp.141-145.

Boulay, B. (1986). Some Difficulties of Learning to Program. *Journal of Educational Computing Research*, 2(1), pp.57-73.

Braffitt, D. (2015). *Radford University - Don Braffitt - ITEC 109 Spring 2015 - Syllabus*. [online] Radford.edu. Available: http://www.radford.edu/~dbraffitt/itec109/2015/spring/ [Accessed 22 Jun. 2016].

Brown, N. C. C., and Altadmri, A. (2014). Investigating Novice Programming Mistakes: Educator Beliefs vs Student Data. In: *ICER '14 International Computing Education Research Conference*, 11-13 August 2014, Glasgow, United Kingdom. pp. 43-50.

Brown, N. C. C., Kölling, M., McCall, D., and Utting, I. (2014). Blackbox: A large scale repository of novice programmers' activity. In: *SIGCSE '14 The 45th ACM Technical Symposium on Computer Science Education*, 05-08 March 2014, Atlanta, GA, USA. pp. 223-228.

Brown, N. (2013). Blackbox: large scale programming education data collection. [Blog] *Academic Computing*. Available from: https://academiccomputing.wordpress.com/2013/08/13/blackbox-large-scale-programming-education-data-collection/ [Accessed 20 Apr. 2015].

Bruce, K. B., Danyluk, A. and Murtagh, T. (2005). Why Structural Recursion Should Be Taught Before Arrays in CS 1. In: Tymann, P. and Baldwin, D. ed. *SIGCSE'05*, *St. Louis, Missouri, USA. February 23–27, 2005*. NY, USA: ACM New York, pp.246-250.

Bruce, K. B. (2004). Controversy on How to Teach CS 1: A discussion on the SIGCSE-members mailing list. In: Walker, H. M. ed. *ITiCSE-WGR 2004*, *Leeds, United Kingdom June 28-30, 2004*. NY, USA: ACM New York, pp.29-34.

Burger, K. R. (2003) Teaching Two-Dimensional Array Concepts in Java With Image Processing Examples. In: Knox, D. and Joyce, D. ed. *The 34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'03)*, *Reno, Nevada, USA. February 19-23, 2003*. NY, USA: ACM New York, pp.205-209.

Chase, R. (2009). *"String index out of range: 0" ...Can someone tell me what's wrong here? (Java)*. [Online] MacRumors Forums. Available from: http://forums.macrumors.com/threads/string-index-out-of-range-0-can-someone-tell-me-whats-wrong-here-java.710415/ [Accessed 12 Sep. 2015].

Chien, C. and Haddad, A. (2015). *SYLLABUS Introduction to Object Oriented Programming (C. Chien)*. [Online] Users.rowan.edu. Available from: http://users.rowan.edu/~chien/IOOP_Syllabus_Spring16.html [Accessed 11 Apr. 2016].

Clark, D., MacNish, C. and Royle, G. F. (1998). Java as a Teaching Language - Opportunities, Pitfalls and Solutions. In: Strooper, P. ed. *3rd Australasian Conference on Computer Science Education (ACSE '98)*, *The University of Queensland, Bisbane, Queensland, Australia July 8-10, 1998*. NY, USA: ACM New York, pp.173-179.

Cloudbings.com, (2015). *java.lang.ArrayIndexOutOfBoundsException: 2 >= 0*. [Online] Available from: http://cloudbings.com/questions/3869259/java-lang-arrayindexoutofboundsexception-2-0 [Accessed 26 Aug. 2015].

Coderanch.com, (2013). *Coordinate out of bounds error when manipulating pixels* [Solved] (Beginning Java forum at Coderanch). [Online] Available from: http://www.coderanch.com/t/586762/java/java/Coordinate-bounds-error-manipulating-pixels [Accessed 15 Aug. 2015].

Collins, J. (2003). *Comparison of Relational and Multi-Dimensional Database Structures*. [online] Alphadevx.com. Available at: http://www.alphadevx.com/a/36-Comparison-of-Relational-and-Multi-Dimensional-Database-Structures [Accessed 1 Jun. 2016].

Community.oracle.com, (2007). *Teaching myself | Oracle Community*. [Online] Community.oracle.com. Available from: https://community.oracle.com/thread/1275649 [Accessed 4 Sep. 2015].

Community.oracle.com, (2006). *Array Index Out Of BoundsException? | Oracle Community*. [Online] Available from: https://community.oracle.com/thread/2115854?start=0&tstart=0 [Accessed 04 Aug. 2015].

Community.oracle.com, (2002). *Exception Exception IndexOutOfBoundsException using Vector. Christmas homework! | Oracle Community*. [Online] Community.oracle.com. Available from: https://community.oracle.com/thread/1157235 [Accessed 20 Aug. 2015].

COS 126: Precepts (Spring 2016), (2016). PRECEPTS. [Online] Cs.princeton.edu. Available from: http://www.cs.princeton.edu/courses/archive/spring16/cos126/precepts.html [Accessed 11 Apr. 2016].

Cprogramming.com, (2006). *C++ constructs*. [Online] C Board. Available from: http://cboard.cprogramming.com/cplusplus-programming/79546-cplusplus-constructs.html [Accessed 19 Feb. 2015].

Dale, N. (2006). Most difficult topics in CS1. *ACM SIGCSE Bulletin*, 38(2), pp.49-53.

Deitel, P., Deitel, P. and Deitel, H. (2012). *Java How to Program, 9/e*. Upper Saddle River, New Jersey, USA: Prentice Hall.

Donchev, I. and Todorova, E. (2008). Object-Oriented Programming in Bulgarian Universities' Informatics and Computer Science Curricula. *Informatics in Education*, 7(2), pp.159-172.

Dotnetperls.com, (2014). *Java Array Examples and Performance*. [Online] Dotnetperls.com. Available from: http://www.dotnetperls.com/array-java [Accessed 8 Feb. 2016].

Dreamincode.net, (2011). *ArrayIndexOutOfBoundsException Error [solved] - Java | Dream.In.Code*. [Online] Available from: http://www.dreamincode.net/forums/topic/222136-arrayindexoutofboundsexception-error-solved/ [Accessed 08 Aug. 2015].

Drysdale, S. (2015). *CS 10: Winter 2016 Schedule*. [Online] Cs.dartmouth.edu. Available from: http://www.cs.dartmouth.edu/~scot/cs10/schedule.html [Accessed 11 Apr. 2016].

Dupont, T. (2015). *C11 - Introduction to Computer Science*. 1st edn. Appleton, Wisconsin, USA: St. Francis Xavier Catholic School System, pp.1-4.

Ebrahimi, A. (1994). Novice programmer errors: language constructs and plan composition. International Journal of Human-Computer Studies, 41(4), pp.457-480.

Eckel, B. (2006). *Thinking in Java (4th Edition)*. Upper Saddle River, New Jersey, USA: Prentice Hall.

Ehlert, A. and Schulte, C. (2009). Empirical comparison of objects-first and objects-later. In: Clancy, M., Caspersen, M. and Lister, R. ed. *International Computing Education Research Workshop (ICER'09)*, *Berkeley, California, USA. August 10–11, 2009*. New York, USA: Association for Computing Machinery (ACM), pp.15-26.

Ehlert, A. and Schulte, C. (2007). Learners' Views on Objects-First and Objects-Later — Results of an Exploratory Study. Position paper presented at Eleventh Workshop on Pedagogies and Tools for the Teaching and Learning of Object-Oriented Concepts, *ECOOP 2007*, Berlin, Germany. July 30, 2007. [no pagination].

Farrell, J. (2013). *An Object-Oriented Approach to Programming Logic and Design*. 4th edn. Boston, Massachusetts, USA: Cengage Learning.

FathomEnthusiast, (2011). *Simple Linear Regression - Intercept and Slope Coefficients in SPSS*. [Online] FathomEnthusiast. Available from: https://www.youtube.com/watch?v=m44pY-DWBd8 [Accessed 14 Mar. 2016].

Fisler, K., (2014) The Recurring Rainfall Problem. In: Cutts, Q., Simon, B. and Dorn, B., ed. *International Computing Education Research Conference (ICER '14), Glasgow, Scotland, UK. August 11-13*. NY, USA: ACM New York, pp.35-42.

Garbinato, B. (2015). *Programmation par objets*. [online] Hec.unil.ch/hec/home. Available at: https://hec.unil.ch/hec/syllabus/descriptif/1964?dyn_lang=en [Accessed 22 Jun. 2016].

GREEN, T. (1977). Conditional program statements and their comprehensibility to professional programmers. *Journal of Occupational Psychology*, 50(2), pp.93-109.

Guzdial, M. (2015), *Learner Centered Design of Computing*, San Rafael, California, USA: Morgan & Claypool Publishers.

Guzdial, M. and Ericson, B. (2007). *Introduction to Computing and Programming with Java: A Multimedia Approach*. Upper Saddle River, N.J.: Pearson Prentice Hall.

Hanks, B. and Brandt, M. (2009). Successful and Unsuccessful Problem Solving Approaches of Novice Programmers. In: Lewandowski, G. and Wolfman, S. ed. *SIGCSE'09*, *Chattanooga, Tennessee, USA March 3–7, 2009*. NY, USA: ACM New York, pp.24-28.

Haziza, D., Chauvet, G. and Deville, J. (2010). SAMPLING AND ESTIMATION IN THE PRESENCE OF CUT-OFF SAMPLING. *Australian & New Zealand Journal of Statistics*, 52(3), pp.303-319.

Hazzan, O., Lapidot, T. and Ragonis, N. (2011). *Guide to teaching computer science*. London: Springer.

Holden, E. and Weeden, E. (2004) The Experience Factor in Early Programming Education. In: Eydie Lawson, ed. *Special Interest Group on Information Technology Education (SIGITE'04)*, *Salt Lake City, Utah, USA. October 28-30, 2004*. NY, USA: ACM New York, pp.211-218.

Horstmann, C. (2012). *Big Java: Late Objects*. Hoboken, New Jersey, USA: J. Wiley & Sons.

Horstmann, C. (2009). *Big Java, 4th Edition*. Hoboken, New Jersey, USA: John Wiley & Sons.

Howe, E., Thornton, M., and Weide, B., W. (2004). Components-first approaches to CS1/CS2: principles and practice. In: Dann, W. and Naps, T. ed. *Special Interest Group on Computer Science Education (SIGCSE'04)*, *Norfolk, Virginia, USA. March 3–7, 2004*. NY, USA: ACM New York, pp.291-295.

Hyland, E. and Clinch, G. (2002). Initial Experiences Gained and Initiatives employed in the Teaching of Java Programming in the Institute of Technology Tallaght. In: Power, J. ed. *PPPJ'02/IRE'02*, *Trinity College, Dublin, Ireland June 13-14, 2002*. Ireland: National University of Ireland Maynooth, County Kildare, Ireland, pp.101-106.

Jadud, M. C., and Dorn, B. (2015). Aggregate Compilation Behavior: Findings and Implications from 27,698 Users. In: Sheard, J. and Cutts, Q. ed. *International Conference on International Computing Education Research (ICER'15)*, *Omaha, NE, USA. August 09-13, 2015*. NY, USA: ACM New York, pp.131-139.

Jalayer Academy, (2013a). *Excel - Time Series Forecasting - Part 2 of 3*. [online] YouTube. Available at: https://www.youtube.com/watch?v=5C012eMSeIU [Accessed 8 Mar. 2016].

Jalayer Academy, (2013b). *Excel - Time Series Forecasting - Part 3 of 3*. [online] YouTube. Available at: https://www.youtube.com/watch?v=kcfiu-f88JQ&feature=iv&src_vid=5C012eMSeIU&annotation_id=annotation_944828 [Accessed 8 Mar. 2016].

Javaprogrammingforums.com, (2014). *String index out of range error*. [Online] Javaprogrammingforums.com. Available from: http://www.javaprogrammingforums.com/whats-wrong-my-code/41183-string-index-out-range-error.html [Accessed 10 Sep. 2015].

Java Tutorial Online, (2010). *Substring Index Out of Range or StringIndexOutOfBoundsException Java*. [Blog] Java Tutorial Online. Available from: http://javacodeonline.blogspot.co.uk/2010/12/substring-index-out-of-range-or.html [Accessed 13 Sep. 2015].

Java With Us, (2013). *Array of Objects - Java Tutorial - Java With Us*. [Online] Javawithus.com. Available from: http://www.javawithus.com/tutorial/array-of-objects [Accessed 23 May 2016].

Java2s.com, (2002). *Java's Unchecked RuntimeException Subclasses : Exception « Development « Java Tutorial*. [Online] Available from: http://www.java2s.com/Tutorial/Java/0120__Development/JavasUncheckedRuntimeExceptionSubclasses.htm [Accessed 7 Dec. 2015].

Kölling, M. (2015). Lessons from the Design of Three Educational Programming Environments: Blue, BlueJ and Greenfoot. *International Journal of People-Oriented Programming*, 4(1), pp.5-32.

Kölling, M. (2012). Project Blackbox – Better data for programming education research. [Blog] *mik's blog Thoughts on Programming Education*. Available from: http://blogs.kent.ac.uk/mik/2012/07/blackbox/ [Accessed 23 Apr. 2015].

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

Kölling, M. (1999a). The problem of teaching object-oriented programming Part I: Languages. *Journal of Object-Oriented Programming*, 11(8), pp.8-15.

Kölling, M. (1999b). The problem of teaching object-oriented programming Part II: *Environment*. *Journal of Object-Oriented Programming*, 11(9). pp. 6-12.

Kuittinen, M. and Sajaniemi, J. (2004). Teaching Roles of Variables in Elementary Programming Courses. In: Martyn Clark and Amruth Kumar, ed. ITiCSE '04 , Leeds, United Kingdom June 28–30, 2004. NY, USA: ACM New York, pp.57-61.

Larsen, F. (2012). *VERY simple: String index out of bounds exception*. [Online] Physics Forums - The Fusion of Science and Community. Available from: https://www.physicsforums.com/threads/very-simple-string-index-out-of-bounds-exception.619717/ [Accessed 12 Sep. 2015].

Lasseter, J. (2015). CPSC 124. [Online] Math.hws.edu. Available from: http://math.hws.edu/lasseter/teaching/F15/CPSC124/calendar.html [Accessed 11 Apr. 2016].

Latham, J. (2015). *Undergraduate Syllabus (School of Computer Science - The University of Manchester).* [Online] Studentnet.cs.manchester.ac.uk. Available from: http://studentnet.cs.manchester.ac.uk/ugt/COMP16121/syllabus/ [Accessed 2 May 2016].

Lentzsch, K. (2006). *Java Source: CellConstraints*. [Online] Jexamples.com. Available from: http://www.jexamples.com/vSrc/5439/com.jgoodies.forms.layout.CellConstraints?prodId=jgoodies_forms&lineNo=701&implExtId=332123&queryText=java.lang.IndexOutOfBoundsException.new&qType=clsMeth#L699 [Accessed 22 Sep. 2015].

Lewandowski, S. (n.d.). *Design Issues in Java and C++*. 1st ed. [ebook] Providence, Rhode Island, New England, USA: Brown University, pp.1-28. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.1547&rep=rep1&type=pdf [Accessed 28 Apr. 2016].

Lost In Transition, (2009). BigDecimal and "java.lang.ArithmeticException: Non-terminating decimal expansion". [Online] Lost In Transition. Available from: https://jaydeepm.wordpress.com/2009/06/04/bigdecimal-and-non-terminating-decimal-expansion-error/ [Accessed 25 Sep. 2015].

Lyman-Abramovitch, M., Higuera, J. and Tremblay, J. (2014). *COMP-202A: Foundations of Programming*. 1st edn. [ebook] Montreal, Québec: McGill University, pp.1-10. Available from: http://cs.mcgill.ca/~cs202/2014-09/web/Sylabus%20-%20fall%202014.pdf [Accessed 3 May 2016].

Maneas, S. (2013). *java.lang.arrayindexoutofboundsexception – How to handle Array Index Out Of Bounds Exception*. [Online] Examples Java Code Geeks. Available from: https://examples.javacodegeeks.com/java-basics/exceptions/java-lang-arrayindexoutofboundsexception-how-to-handle-array-index-out-of-bounds-exception/ [Accessed 21 Sep. 2015].

Markov, Z. (2010). CS501 Syllabus. [online] cs.ccsu.edu. Available at: http://www.cs.ccsu.edu/~markov/ccsu_courses/501syllabus.html [Accessed 22 Jun. 2016].

McConnell, J. and Burhans, D. (2002). The evolution of CS1 textbooks. *32nd Annual Frontiers in Education*, 1, pp.T4G-1 - T4G-6 vol.1.

McSweeney, J. D. (2015). *Course Syllabus CISP 2660 Java Programming.* 1st edn. [ebook]. Fayetteville, Tennessee, USA: Motlow State Community College, pp.1-5. Available from: http://www.mscc.edu/business/syllabi/CISP_2660.pdf [Accessed 2 May 2016].

Meisalo, V., Sutinen, E. and Torvinen, S. (2003) CHOOSING APPROPRIATE METHODS FOR EVALUATING AND IMPROVING THE LEARNING PROCESS IN DISTANCE PROGRAMMING COURSES. In: EP Innovations, Inc., ed. *33rd ASEE/IEEE Frontiers in Education Conference*, *Boulder, Colorado, USA November 5 - 8, 2003*. Champaign, IL.: Stipes Publishing L.L.C., pp.T2B-11-T2B-16.

Minitorn.tlu.ee, (n.d.). : Class Canvas3D. [Online] Minitorn.tlu.ee. Available from: http://minitorn.tlu.ee/~jaagup/kool/java/abiinfo/3d/html/javax/media/j3d/Canvas3D.html [Accessed 5 Sep. 2015].

Nathan, M. J., Koedinger, K. R., and Alibali, M. W., (2001) Expert Blind Spot: When Content Knowledge Eclipses Pedagogical Content Knowledge. In: L. Chen et al., ed. *The 3rd ICCS International Conference on Cognitive Science*, *Beijing, China August 27—31*. Hefei, Anhui, China: USTC Press, pp.644–648.

Nieminen, J. (2008). *Bubble sort misconceptions*. [Online] Warp.povusers.org. Available from: http://warp.povusers.org/grrr/bubblesort_misconceptions.html [Accessed 20 Apr. 2016].

Nieminen, J. (2006). *Bubble sort as the first sorting algorithm*. [Online] Warp.povusers.org. Available from: http://warp.povusers.org/grrr/bubblesort_eng.html [Accessed 20 Apr. 2016].

Nilsson, E. (2016). *TDDC30 Programming in Java, Data Structures and Algorithms*. [Online] Linköping University. Available from: https://www.ida.liu.se/~TDDC30/current/lectures/ [Accessed 3 May 2016].

Objectmix.com, (2007). *Copying JPanel components to another JPanel*. [Online] Available from: http://objectmix.com/java/100961-copying-jpanel-components-another-jpanel.html [Accessed 26 Aug. 2015].

O'Brien, K. (2016). San Jose State University CS 46A - Introduction to Programming Section 1. 1st edn. [ebook] San Jose, CA, USA: DEPARTMENT OF COMPUTER SCIENCE SAN JOSÉ STATE UNIVERSITY | COLLEGE OF SCIENCE, pp.1-9. Available from: http://www.sjsu.edu/cs/practicalities/syllabi/spring-2016/CS46A-Section1-S16.pdf [Accessed 17 May 2016].

Pasa Uysal, M. (2012). The Effects of Objects-First and Objects-Late Methods on Achievements of OOP Learners. *JSEA*, 05(10), pp.816-822.

Pont, M. (1998). Why Java is dangerous. *IEEE Software*, 15(1), pp.20-22.

Pritchard, D. (2015). Frequency Distribution of Error Messages. In: *The Sixth Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at SPLASH 2015*, *Pittsburgh, Pennsylvania, USA. October 25–30, 2015*, pp.1-8.

Project Nayuki, (2015). *Gauss-Jordan elimination over any field (Java)*. [Online] Nayuki.io. Available from: https://www.nayuki.io/res/gauss-jordan-elimination-over-any-field-java/Matrix.java [Accessed 21 Sep. 2015].

Putnam, R., Sleeman, D., Baxter, J. and Kuspa, L. (1986). A Summary of Misconceptions of High School BASIC Programmers. *Journal of Educational Computing Research*, 2(4), pp.459-472.

Reges, S. (2006). Back to basics in CS1 and CS2. In: Baldwin, D., Tymann, P., Haller, S. and Russell, I. ed. *Special Interest Group on Computer Science Education (SIGCSE'06)*, *Houston, Texas, USA. March 1-5, 2006*. NY, USA: ACM New York, pp.293-297.

Robins, A. (2010). Learning edge momentum: a new account of outcomes in CS1. *Computer Science Education*, 20(1), pp.37-71.

Robins, A., Haden, P., and Garner, S. (2006). Problem distributions in a CS1 course. In: Tolhurst, D. and Mann, S. ed. *Eighth Australasian Computing Education Conference*, *Hobart, Tasmania, Australia. January 2006*. Darlinghurst, Australia: Australian Computer Society, Inc., pp.165-173.

Robins, A., Rountree, J. and Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2), pp.137-172.

Sayers, H. M., Nicell, M. A, and Hagan, S. J. (2003). TEACHING JAVA PROGRAMMING: DETERMINING THE NEEDS OF FIRST YEAR STUDENTS. In: *The 4th Annual Conference of the LTSN Subject Centre for Information and Computer Sciences (4th Annual LTSN-ICS Conference)*, *National University of Ireland (NUI), Galway, Ireland. 26 - 28th August 2003*, pp.106-110.

Scott, A., Watkins, M. and Mcphee, D. (2007). *A Step Back from Coding-An Online Environment and pedagogy for Novice Programmers*. 1st ed. [ebook] Trefforest, Wales, UK: Faculty of Advanced Technology-University of Glamorgan, pp.1-7. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.8873&rep=rep1&type=pdf [Accessed 28 Apr. 2016].

Sedgewick, R. and Wayne, K. (2016). *Arrays*. [Online] Introcs.cs.princeton.edu. Available from: http://introcs.cs.princeton.edu/java/14array/ [Accessed 22 May 2016].

Sheller, C. (2015). Data Structures CSC-153 Spring 2016. 1st edn. [ebook] Cedar Rapids, Iowa, USA: Kirkwood Community College, pp.1-6. Available from: http://faculty.kirkwood.edu/pdf/uploaded/262/cs2/sp16/dsmwsp16.pdf [Accessed 13 Apr. 2016].

Shkolnik, A. (2016). *Introduction to Java Programming*. 1st edn. [ebook] Beersheba, Israel: Ben-GurionUniversity of the Negev, pp.1-3. Available from: https://www.cs.bgu.ac.il/~ipc131/wiki.files/Syllabus_IPJ_131.pdf [Accessed 2 May 2016].

Siegfried, R.M., Chays, D. and Herbert, K. (2008) Will There Ever Be Consensus on CS1? In: Arabnia, H.R., Clincy, V.A. and Tadayon, N. ed. *Proceedings of the 2008 International Conference on Frontiers in Education: Computer Science & Computer Engineering, (FECS 2008)*, *Las Vegas, Nevada, USA July 14-17, 2008*. Georgia, USA: CSREA Press, pp.18-23.

Simon, B., Kinnunen, P., Porter, L. and Zazkis, D. (2010) Experience Report: CS1 for Majors with Media Computation. In: Cary Laxer, ed. *The 15th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'10)*, *Bilkent, Ankara, Turkey June 26-30, 2010*. NY, USA: ACM New York, pp.214-218.

Slee, S. (2006). *Computer Science 1 : Syllabus*. [Online] Cs.duke.edu. Available from: https://www.cs.duke.edu/courses/cps001/summer06/syllabus.html [Accessed 20 Apr. 2016].

Smolarski, D. (2003). A first course in computer science: languages and goals. Teaching Mathematics and Computer Science, 1(1), pp.137-152.

Soloway, E., Bonar, J. and Ehrlich, K. (1983). Cognitive strategies and looping constructs: an empirical study. *Communications of the ACM*, 26(11), pp.853-860.

Spohrer, J. and Soloway, E. (1986). Novice mistakes: are the folk wisdoms correct?. *Communications of the ACM*, 29(7), pp.624-632.

Sprague, N. and Chao, A. (2016). CS139 Schedule. [online] W3.cs.jmu.edu. Available at: https://w3.cs.jmu.edu/spragunr/CS139/schedule.shtml [Accessed 22 Jun. 2016].

Srinivasan, K. (2014). *java.lang.ArrayIndexOutOfBoundsException*. [Online] JavaBeat. Available from: http://www.javabeat.net/java-lang-arrayindexoutofboundsexception/ [Accessed 06 Aug. 2015].

Stackoverflow.com, (2014). *Is the row passed to getTableCellRendererComponent supposed to be in bounds?*. [Online] Stackoverflow.com. Available from: http://stackoverflow.com/questions/22931243/is-the-row-passed-to-gettablecellrenderercomponent-supposed-to-be-in-bounds [Accessed 21 Sep. 2015].

Stackoverflow.com, (2011a). *java.lang.ArrayIndexOutOfBoundsException: 0 - Array larger than Index?*. [Online] Stackoverflow.com. Available from: http://stackoverflow.com/questions/7869810/java-lang-arrayindexoutofboundsexception-0-array-larger-than-index [Accessed 07 Aug. 2015].

Stackoverflow.com, (2011b). *why is it not catching the arrays out of bound?*. [Online] Stackoverflow.com. Available from: http://stackoverflow.com/questions/7458449/why-is-it-not-catching-the-arrays-out-of-bound [Accessed 31 Aug. 2015].

Stanford University School of Education, (1984). *Pascal and High-School Students: A Study of Misconceptions*. Occasional Report #009. [Online] Stanford, CA: Education Resources Information Center (ERIC), pp.1-34. Available from: http://files.eric.ed.gov/fulltext/ED258552.pdf [Accessed 1 Jul. 2016].

Stepp, M. and Martin, H. (2015). *CSE 142, Winter 2015*. [Online] Courses.cs.washington.edu. Available from: http://courses.cs.washington.edu/courses/cse142/15wi/calendar.shtml [Accessed 24 Mar. 2016].

Sternberg, R. (2014). 3 Good Reasons to Avoid Arrays in Java Interfaces. [Blog] EclipseSource DEVELOPER. Available from: http://eclipsesource.com/blogs/2014/04/11/3-good-reasons-to-avoid-arrays-in-java-interfaces/ [Accessed 22 May 2016].

Stevens, P. and McCall, D. (2012). What is the Blackbox Data Collection?. [Blog] *Black Box Data Collection*. Available from: https://blackboxdc.wordpress.com/2012/05/21/hello-world/ [Accessed 23 Apr. 2015].

Sullivan, D. (2016). *S-111 — Syllabus*. [Online] Sites.fas.harvard.edu. Available from: http://sites.fas.harvard.edu/~libs111/syllabus.html [Accessed 3 May 2016].

Tew, A. E. (2010). *Assessing fundamental introductory computing concept knowledge in a language independent manner*. Doctor of Philosophy (PhD). Georgia Institute of Technology.

Tew, A. E., McCracken, W. M., and Guzdial, M. (2005). Impact of Alternative Introductory Courses on Programming Concept Understanding. In: Anderson, R., Fincher S. A. and Guzdial, M. ed. *International Computing Education Research Workshop (ICER'05), Seattle, Washington, USA. October 1–2, 2005*. NY, USA: ACM New York, pp.25-35.

Thompson, S. M. (2006). *An Exploratory Study of Novice Programming Experiences and Errors*. MASTER OF SCIENCE Thesis, University of Victoria.

usadye.ru, (2015). *Java Jar Arguments Using Command-Line Arguments in Java- usadye.ru.* [Online] Usadye.ru. Available from: http://usadye.ru/full-desc/aHR0cDovL2phdmEuYWJvdXQuY29tL29kL2phdmFzeW50YXgvYS9Vc2luZy1Db21tYW5kLUxpbmUtQXJndW1lbnRzLmh0bXx8SmF2YSBKYXIgQXJndW1lbnRzfHwwfHxXZWJ8fDI [Accessed 24 May 2016].

Utting, I., Brown, N., Kölling, M., McCall, D. and Stevens, P. (2012). Web-scale Data Gathering with BlueJ. In: Clear, A., Sanders, K. and Simon, B. ed. *The ninth annual International Computing Education Research Conference, (ICER 2012)*, Auckland, New Zealand. September 10 - 12, 2012. NY, USA: ACM New York, pp.1-4.

Ventura, P., Egert, C. and Decker, A. (2004) Ancestor Worship in CS1: On the Primacy of Arrays. In: Schmidt, D. and Cohen, G. ed. *Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'04)*, *Vancouver, British Columbia, Canada. Oct. 24–28, 2004*. NY, USA: ACM New York, pp.68-72.

voidException, (2006). *Java ArrayIndexOutOfBoundsException example - Array index out of bounds exception causes and fixes - Java Tutorials*. [Online] Available from: http://voidexception.weebly.com/array-index-out-of-bounds-exception.html [Accessed 05 Aug. 2015].

Wahls, T. (2014). *COMP 131: Introduction to Computer Science I.* [Online] Users.dickinson.edu. Available from: http://users.dickinson.edu/~wahlst/131/ [Accessed 11 Apr. 2016].

Wallace, R. (2016). *CS161/162/260 - CSI/II & Data Structures*. [Online] Cs.bluecc.edu. Available from: http://cs.bluecc.edu/java/ [Accessed 24 May 2016].

Williams, K. (2016). *GEEN163 Introduction to Computer Programming*. [Online] Williams.comp.ncat.edu. Available from: http://williams.comp.ncat.edu/GEEN163/Index.htm [Accessed 24 Mar. 2016].

Williams, P. (2009). *StringBuffer/StringBuilder question - unexpected results (Java in General forum at Coderanch)*. [Online] Coderanch.com. Available from: http://www.coderanch.com/t/386325/java/java/StringBuffer-StringBuilder-unexpected-results [Accessed 18 Sep. 2015].

Wittman, B., Mathur, A. and Korb, T. (2013). *Start Concurrent: An Introduction to Problem Solving in Java with a Focus on Concurrency*. [West Lafayette, IN]: Purdue University.

www.tutorialspoint.com, (2009). *Java Built-in Exceptions*. [Online] Available from: http://www.tutorialspoint.com/java/java_builtin_exceptions.htm [Accessed 7 Dec. 2015].

www.tutorialspoint.com, (n.d.). *Java.math.BigDecimal.scale() Method*. [Online] www.tutorialspoint.com. Available from: http://www.tutorialspoint.com/java/math/bigdecimal_scale.htm [Accessed 25 Sep. 2015].

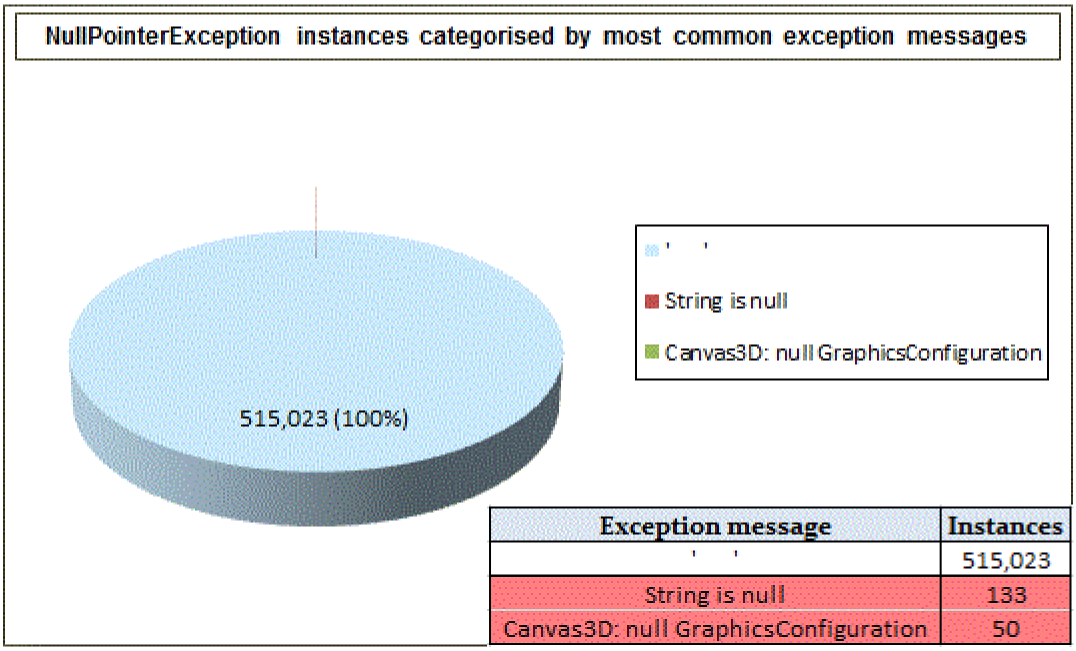
APPENDIXES

APPENDIX A – 26 months 'Total Number' of Error Instances for the 14 preliminary evaluated Runtime Exception classes (Total Numbers here are still unfiltered from non-standard Java library cases, and from users having only 1 session interaction with BlueJ)

Dates	ArrayIndexOutOfBounds	NullPointer	StringIndexOutOfBounds	IndexOutOfBounds	Arithmetic	IllegalArgument	NoSuchElement	Runtime	ClassCast	IllegalState	NegativeArray	UnsupportedOperation	ArrayStore	Security
Jun-13	558	672	158	36	848	59	54	56	54	4		4		
Jul-13	2,243	1,989	908	329	973	162	153	20	32	37	1	5	3	2
Aug-13	2,737	2,703	1,268	366	1,267	193	460	65	34	105	14	3	2	
Sep-13	5,519	7,838	3,134	1,095	1,992	794	738	156	163	134	17	11	2	
Oct-13	17,716	20,257	12,013	3,014	3,540	2,478	2,404	536	474	472	53	41	49	3
Nov-13	28,253	26,029	9,494	4,138	2,738	2,500	2,667	781	672	460	167	355	50	
Dec-13	23,410	24,510	6,350	4,151	2,124	2,037	1,562	1,284	634	312	141	30	29	3
Jan-14	19,038	19,329	5,105	4,234	2,151	2,107	1,413	689	490	293	92	20	43	
Feb-14	20,917	19,608	7,289	3,952	2,333	2,375	2,252	911	782	433	99	77	61	
Mar-14	22,884	25,635	9,062	4,545	2,291	3,271	3,007	534	843	642	128	33	53	
Apr-14	23,386	24,988	6,875	4,109	1,747	2,271	2,639	522	712	658	143	18	98	
May-14	18,233	25,604	4,997	3,194	1,494	1,985	2,097	755	777	531	125	45	53	4
Jun-14	10,494	14,898	3,467	1,735	1,288	1,159	1,074	315	536	127	66	9	18	20
Jul-14	7,078	7,100	2,570	802	1,528	2,456	498	140	275	192	18	20	10	
Aug-14	9,750	7,664	3,650	762	1,459	692	849	168	232	203	44	47	22	
Sep-14	16,514	16,791	8,576	1,775	3,332	1,471	1,651	242	480	419	91	37	27	4
Oct-14	27,246	28,643	18,974	4,624	4,308	2,755	3,151	997	789	867	116	68	56	
Nov-14	40,783	35,942	14,722	6,521	3,810	3,427	4,005	1,142	1,000	850	245	79	48	6
Dec-14	36,026	31,531	9,400	5,760	2,440	2,563	2,477	1,201	1,070	619	246	115	29	
Jan-15	32,517	24,960	9,503	5,990	2,620	2,451	1,960	1,398	829	395	150	61	29	
Feb-15	34,203	25,850	10,902	6,606	2,609	2,503	2,630	1,359	837	360	143	61	45	2
Mar-15	36,960	31,565	9,424	5,707	2,677	3,066	3,192	1,914	1,011	871	146	69	37	
Apr-15	34,796	32,390	6,827	6,515	2,637	4,006	3,027	1,652	884	710	220	115	55	
May-15	26,427	32,942	5,474	4,511	2,123	2,660	2,559	1,036	1,051	557	106	101	38	9
Jun-15	15,974	18,437	5,333	2,068	1,593	1,431	1,822	418	637	352	55	47	23	
Jul-15	9,439	8,238	3,638	954	1,427	726	715	283	473	282	48	34	16	2
Total	**523,101**	**516,113**	**179,113**	**87,493**	**57,349**	**51,598**	**49,056**	**18,574**	**15,771**	**10,885**	**2,674**	**1,505**	**896**	**55**

APPENDIX B –

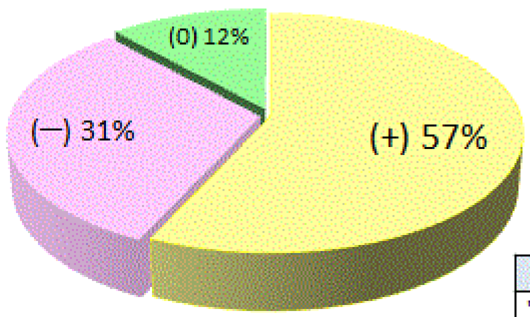
NULLPOINTEREXCEPTION, STRINGINDEXOUTOFBOUNDSEXCEPTION, INDEXOUTOFBOUNDSEXCEPTION and ARITHMETICEXCEPTION classes categorized by their most common exception message types[40] (*Classes' categories of exception-message types, which were apprehended to be deriving from non-standard Java libraries, had been termed ('other message type' - marked as red in graphs below) and their instances had been discarded)



Exception message	Instances
' '	515,023
String is null	133
Canvas3D: null GraphicsConfiguration	50

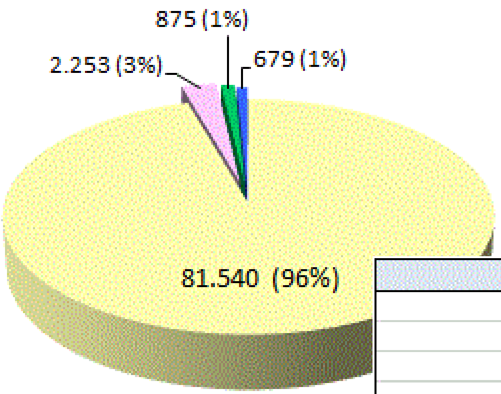
[40] Further information for the categories of 'Exception Message' types of the class:
ARRAYINDEXOUTOFBOUNDSEXCEPTION: Cloudbings.com (2015), Srinivasan (2014), Coderanch.com (2013), Stackoverflow.com (2011a; 2011b), Dreamincode.net (2011), Akraju (2008), Objectmix.com (2007), Community.oracle.com (2006; 2002) and voidException (2006).
NULLPOINTEREXCEPTION: Community.oracle.com (2007) and Minitorn.tlu.ee (n.d.).
STRINGINDEXOUTOFBOUNDSEXCEPTION: amin (2014), Javaprogrammingforums.com (2014), Larsen (2012), Java Tutorial Online (2010), Chase (2009) and Williams (2009).
INDEXOUTOFBOUNDSEXCEPTION: Project Nayuki (2015), Stackoverflow.com (2014), Maneas (2013), Alexander (2010) and Lentzsch (2006).
ARITHMETICEXCEPTION: Lost In Transition (2009) and www.tutorialspoint.com (n.d.).

StringIndexOutOfBoundsException instances categorised by most common exception messages

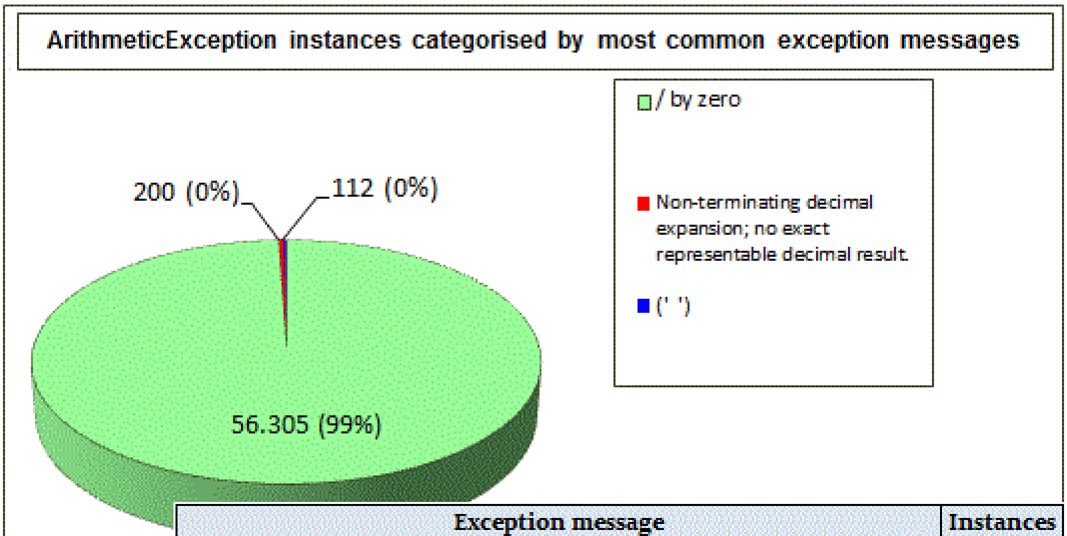


Exception message	Instances
'String index out of range: +'	101,608
'String index out of range: –'	56,353
'String index out of range: 0'	20,825
start > length()	126
other message type	201

IndexOutOfBoundsException instances categorised by most common exception messages



Exception message	Instances
Index: , Size:	81,540
row|col is out of bounds	2,253
' '	875
(,)	679
must be between	154
npoints > xpoints.length || npoints > ypoints.length	122
other message type	1,870



Exception message	Instances
/ by zero	56,305
Non-terminating decimal expansion; no exact representable decimal result.	200
(' ')	112
other message type	732

APPENDIX C – The ArrayIndexOutOfBoundsException class's 'Total Number' and 'Users' Number (derived from the class's commonest message type categories errors of **Figure 2**, and from the Users who had more than one session interaction with BlueJ), as well as its weekly 'Average Numbers of Errors (per-user)' (Average Number = Total Number / Users Number) values.

ArrayIndexOutOfBoundsException				ArrayIndexOutOfBoundsException			
Week	Total	Users	Average Errors (per-user)	Week	Total	Users	Average Errors (per-user)
1	48,784	15,186	3.212	57	709	243	2.918
2	23,701	6,849	3.461	58	704	208	3.385
3	20,707	6,221	3.329	59	683	217	3.147
4	22,043	6,334	3.480	60	586	201	2.915
5	20,940	6,168	3.395	61	976	235	4.153
6	22,492	6,306	3.567	62	626	186	3.366
7	22,301	6,140	3.632	63	651	202	3.223
8	20,784	5,802	3.582	64	719	216	3.329
9	21,448	5,966	3.595	65	896	186	4.817
10	21,059	5,818	3.620	66	679	175	3.880
11	19,630	5,500	3.569	67	460	136	3.382
12	19,417	5,330	3.643	68	423	142	2.979
13	17,753	5,035	3.526	69	480	138	3.478
14	16,141	4,321	3.735	70	478	116	4.121
15	13,099	3,775	3.470	71	481	121	3.975
16	10,792	3,196	3.377	72	398	115	3.461
17	8,932	2,646	3.376	73	350	104	3.365
18	8,177	2,489	3.285	74	418	94	4.447
19	8,620	2,482	3.473	75	335	109	3.073
20	7,976	2,284	3.492	76	279	98	2.847
21	7,581	2,220	3.415	77	327	84	3.893
22	6,507	2,075	3.136	78	525	103	5.097
23	7,071	2,043	3.461	79	347	78	4.449
24	6,724	2,042	3.293	80	323	81	3.988
25	7,312	2,034	3.595	81	246	73	3.370
26	6,906	1,934	3.571	82	299	70	4.271
27	7,034	1,953	3.602	83	274	71	3.859
28	5,553	1,717	3.234	84	219	70	3.129
29	5,535	1,585	3.492	85	314	79	3.975
30	4,643	1,382	3.360	86	306	55	5.564
31	4,217	1,250	3.374	87	264	52	5.077
32	4,532	1,294	3.502	88	226	58	3.897
33	4,025	1,227	3.280	89	232	46	5.043
34	3,733	1,028	3.631	90	164	43	3.814
35	2,833	861	3.290	91	192	27	7.111
36	2,899	811	3.575	92	185	38	4.868
37	2,296	717	3.202	93	133	32	4.156
38	2,124	691	3.074	94	85	26	3.269
39	2,002	620	3.229	95	75	24	3.125
40	1,720	520	3.308	96	118	17	6.941
41	1,748	426	4.103	97	130	19	6.842
42	1,486	436	3.408	98	38	14	2.714
43	1,208	378	3.196	99	38	12	3.167
44	1,081	295	3.664	100	104	19	5.474
45	1,023	297	3.444	101	72	12	6.000
46	796	280	2.843	102	43	9	4.778
47	792	285	2.779	103	64	13	4.923
48	733	259	2.830	104	2	1	2.000
49	812	263	3.087	105	16	5	3.200
50	870	251	3.466	106	8	4	2.000
51	928	278	3.338	107	4	3	1.333
52	733	244	3.004	108	6	2	3.000
53	860	260	3.308	109	13	4	3.250
54	1,105	259	4.266	110	3	2	1.500
55	731	233	3.137	111	0	0	0.000
56	867	253	3.427	112	0	0	0.000

APPENDIX D – The StringIndexOutOfBoundsException class's 'Total Number' and 'Users' Number (derived from the class's commonest message type categories errors of **APPENDIX B**, and from the Users who had more than one session interaction with BlueJ), as well as its weekly 'Average Numbers of Errors (per-user)' (Average Number = Total Number / Users Number) values.

StringIndexOutOfBoundsException				StringIndexOutOfBoundsException			
Week	Total	Users	Average Errors (per-user)	Week	Total	Users	Average Errors (per-user)
1	18,086	6,792	2.663	57	267	105	2.543
2	9,630	3,556	2.708	58	211	81	2.605
3	10,143	3,531	2.873	59	240	91	2.637
4	10,480	3,461	3.028	60	277	90	3.078
5	11,308	3,474	3.255	61	178	74	2.405
6	10,250	3,226	3.177	62	234	72	3.250
7	9,119	2,900	3.144	63	147	57	2.579
8	8,417	2,853	2.950	64	152	56	2.714
9	7,870	2,611	3.014	65	154	61	2.525
10	7,098	2,343	3.029	66	132	50	2.640
11	6,887	2,163	3.184	67	169	62	2.726
12	5,826	1,884	3.092	68	148	46	3.217
13	4,964	1,588	3.126	69	101	42	2.405
14	4,303	1,428	3.013	70	152	47	3.234
15	3,662	1,286	2.848	71	80	37	2.162
16	2,866	1,011	2.835	72	71	30	2.367
17	2,374	832	2.853	73	118	47	2.511
18	2,403	821	2.927	74	169	40	4.225
19	2,171	709	3.062	75	90	37	2.432
20	1,806	680	2.656	76	98	35	2.800
21	2,280	677	3.368	77	77	25	3.080
22	1,794	604	2.970	78	110	38	2.895
23	1,654	592	2.794	79	81	23	3.522
24	1,624	591	2.748	80	141	30	4.700
25	1,426	532	2.680	81	107	33	3.242
26	1,730	549	3.151	82	83	22	3.773
27	1,720	565	3.044	83	48	17	2.824
28	1,436	506	2.838	84	53	26	2.038
29	1,410	451	3.126	85	60	24	2.500
30	1,366	464	2.944	86	33	14	2.357
31	1,042	391	2.665	87	71	17	4.176
32	913	343	2.662	88	63	15	4.200
33	971	379	2.562	89	33	11	3.000
34	930	339	2.743	90	18	12	1.500
35	982	347	2.830	91	48	6	8.000
36	721	290	2.486	92	87	12	7.250
37	602	244	2.467	93	103	12	8.583
38	742	242	3.066	94	4	3	1.333
39	576	217	2.654	95	6	2	3.000
40	489	192	2.547	96	16	7	2.286
41	611	146	4.185	97	33	7	4.714
42	513	139	3.691	98	19	7	2.714
43	316	124	2.548	99	6	3	2.000
44	392	124	3.161	100	28	8	3.500
45	440	144	3.056	101	13	3	4.333
46	420	115	3.652	102	56	6	9.333
47	312	125	2.496	103	24	5	4.800
48	233	103	2.262	104	7	3	2.333
49	234	101	2.317	105	3	2	1.500
50	233	80	2.913	106	3	3	1.000
51	276	99	2.788	107	0	0	0.000
52	292	96	3.042	108	0	0	0.000
53	310	121	2.562	109	4	2	2.000
54	303	109	2.780	110	0	0	0.000
55	248	102	2.431	111	0	0	0.000
56	299	118	2.534	112	0	0	0.000

APPENDIX E – The IndexOutOfBoundsException class's 'Total Number' and 'Users' Number (derived from the class's commonest message type categories errors of **APPENDIX B**, and from the Users who had more than one session interaction with BlueJ), as well as its weekly 'Average Numbers of Errors (per-user)' (Average Number = Total Number / Users Number) values.

IndexOutOfBoundsException			
Week	**Total**	**Users**	**Average Errors (per-user)**
1	5,968	2,077	2.873
2	3,759	1,156	3.252
3	3,777	1,237	3.053
4	3,601	1,267	2.842
5	3,344	1,229	2.721
6	3,888	1,376	2.826
7	3,700	1,338	2.765
8	3,380	1,177	2.872
9	3,217	1,145	2.810
10	3,005	1,081	2.780
11	3,129	999	3.132
12	2,909	937	3.105
13	3,050	986	3.093
14	2,314	727	3.183
15	2,386	725	3.291
16	2,508	656	3.823
17	1,633	483	3.381
18	1,175	402	2.923
19	1,373	456	3.011
20	1,498	483	3.101
21	1,241	423	2.934
22	1,508	425	3.548
23	1,311	434	3.021
24	1,391	426	3.265
25	1,280	425	3.012
26	1,311	438	2.993
27	1,112	371	2.997
28	1,060	336	3.155
29	1,014	359	2.825
30	1,090	274	3.978
31	728	246	2.959
32	756	268	2.821
33	802	251	3.195
34	823	260	3.165
35	599	188	3.186
36	410	161	2.547
37	491	174	2.822
38	561	184	3.049
39	454	133	3.414
40	365	119	3.067
41	325	89	3.652
42	150	57	2.632
43	296	56	5.286
44	196	67	2.925
45	179	70	2.557
46	149	47	3.170
47	156	44	3.545
48	85	38	2.237
49	74	36	2.056
50	98	27	3.630
51	152	45	3.378
52	103	34	3.029
53	132	39	3.385
54	124	44	2.818
55	128	41	3.122
56	66	26	2.538

IndexOutOfBoundsException			
Week	**Total**	**Users**	**Average Errors (per-user)**
57	56	22	2.545
58	138	44	3.136
59	91	37	2.459
60	128	37	3.459
61	121	36	3.361
62	56	27	2.074
63	96	31	3.097
64	70	38	1.842
65	104	32	3.250
66	78	30	2.600
67	64	24	2.667
68	80	25	3.200
69	76	22	3.455
70	30	12	2.500
71	58	19	3.053
72	41	15	2.733
73	105	27	3.889
74	22	14	1.571
75	38	14	2.714
76	45	18	2.500
77	28	12	2.333
78	56	22	2.545
79	54	22	2.455
80	48	19	2.526
81	61	16	3.813
82	23	8	2.875
83	18	9	2.000
84	47	13	3.615
85	18	8	2.250
86	21	8	2.625
87	42	6	7.000
88	32	8	4.000
89	77	13	5.923
90	47	12	3.917
91	11	7	1.571
92	29	9	3.222
93	28	10	2.800
94	10	4	2.500
95	9	5	1.800
96	10	4	2.500
97	13	4	3.250
98	4	1	4.000
99	3	3	1.000
100	7	3	2.333
101	0	0	0.000
102	7	3	2.333
103	8	1	8.000
104	0	0	0.000
105	0	0	0.000
106	0	0	0.000
107	1	1	1.000
108	0	0	0.000
109	0	0	0.000
110	0	0	0.000
111	0	0	0.000
112	0	0	0.000

APPENDIX F – Pritchard (2015) online Java Blackbox data June-2013/October-2013 (excluding the beginning of June). (Pritchard 2015, no pagination)

25389	java.lang.NullPointerException
21414	non-static <METHOD> cannot be referenced from a static context
20995	'(' expected
20232	java.lang.ArrayIndexOutOfBoundsException
20111	cannot return a value from <METHOD> result type is void
19528	non-static <VARIABLE> cannot be referenced from a static context
19373	<CLASS> is public and should be declared in a file named <FILE>
18898	java.util.InputMismatchException
18782	<NAME> is not abstract and does not override abstract method <METHOD> in <CLASS>
18413	missing <METHOD>, or declare abstract
18229	<TYPE> cannot be dereferenced
18177	unreachable statement
16475	no suitable constructor found for <NAME>
16340	incompatible types - found: <TYPE> but expected <TYPE>
16336	'{' expected
14834	<NAME> has private access in <NAME>
13420	java.lang.StringIndexOutOfBoundsException
11303	incomparable types: <TYPE> and <TYPE>
11239	unclosed character literal
10673	java.lang.NumberFormatException # is an invalid <TYPE>
10464	unreported exception <NAME> thrown
10323	']' expected
8204	'void' type not allowed here
7145	java.lang.ArithmeticException Division by zero.
6861	illegal escape character
6744	unmappable character for encoding <NAME>
6600	integer number too large: #
6453	cannot find symbol - constructor <NAME>
5616	while expected
5388	array required, but <TYPE> found
4977	class expected
4920	bad operand type <TYPE> for unary operator ' '
4267	java.lang.StackOverflowError
3807	inconvertible types, found: <TYPE> required: <TYPE>
3728	array dimension missing
3639	java.lang.IndexOutOfBoundsException