Novice Java Programmers' Perceptions on the Correctness, Code Assessment, Fairness of Assessment, and Bases of Assessment on Codes with Syntax and Logical Errors: A Comparative Perspective

Rex P. Bringula, PhD
College of Computer Studies and Systems
University of the East
2219 C.M. Recto Avenue, Sampaloc, Manila, Philippines

Abstract

This descriptive study determined the perceptions of novice programmers on the correctness, assessment, fairness of assessment, and bases of assessment on codes with syntax and logical errors. It also determined whether these perceptions on codes with these two types of programming errors differed significantly. It was found out that novice programmers considered a program to be somewhat correct regardless of the types of error it had. They gave a passing mark to flawed codes and they agreed to a lesser extent that the teachers' assessment on the codes was fair. It was revealed that a code with syntax error was rated based mainly on the difficulty of the error while a code with logical error was rated based primarily on the effort exerted in doing the code. It was concluded that novice programmers also had the incorrect notions of program assessment and fairness of assessment. One of the four hypotheses was rejected. In the light of the findings and conclusions presented, implications and recommendations were set forth.

Keywords: Java; Fairness of assessment; Logical error; Novice programmer; Program assessment; Program correctness; Syntax error

1. Introduction

The demand for programmers in the Philippines and abroad is apparent. However, there is a low supply of labor force that could meet this demand. Students are initially attracted to take Information Technology-related degree programs (Information Technology, Computer Science, and Information Systems) but later on shifted to another degree program. This is attributed to the anti-social image of the degree (Lenox et al. 2008) and difficulty of the programming subjects (Robins et al., 2003; Pendergast, 2005; Sloan and Troy, 2008).

Understanding novice programmers is the first step in helping them to learn to program. Different studies have been conducted in an attempt to understand novice programmers. Novice programmers lack programming strategies (Carbone et al., 2009). They have difficulties in using selection structure (Gobil et al., 2009), expressing natural language solutions into computer programming language (Ebrahimi, 1994; Kelleher and Pausch, 2005), analyzing and designing mathematical expressions, naming variables and assigning suitable data types and structures to these variables, evaluating correctly output statements, arithmetic expressions, and relationship expressions (Gobil et al., 2009), and debugging loop conditions, conditional logic, arithmetic errors, and data initialization and update (Fitzgerald et al., 2008). One study also attempted to identify the top java errors of novice programmers (Jackson et al., 2005).

Literature also shows that novice programmers have misconceptions of program correctness. Students (i.e., novice programmers) tend to tolerate program errors (Ben-David Kolikant, 2005). They would consider a program correct if the program "worked in general" or output a correct answer for "many input examples" but not necessarily error-free for the input set (Ben-David Kolikant and Mussai, 2008). Flawed programs would be considered partially correct as long as they found a "grain of correctness" in the program (Ben-David Kolikant and Mussai, 2008).

Sponsoring information: This paper is funded by the University of the East.

In other words, novice programmers perceived program correctness as not dichotomous (Ben-David Kolikant, 2005). However, what is lacking in the literature is the distinct discussion regarding the misconceptions of novice programmers on the two types of programming errors – syntax and logical errors. In the light of this research gap, this study has been conceived. This study aimed to explain further the misconceptions of novice programmers that arose from these errors. It attempted to determine whether the perceptions of novice programmers on the correctness, assessment, fairness of assessment, and bases of assessment on codes with syntax and logical errors differed significantly.

Toward these aims, it sought answers to the following questions.

1. How do the respondents perceive the correctness of a code with syntax and logical errors?
2. How do the respondents assess a code with syntax and logical errors?
3. How do the respondents perceive the fairness on the assessment of the teachers on a code with syntax and logical errors?
4. What are the bases of the respondents in assessing a code with syntax and logical errors?
5. Is there a significant difference between the
    a. perceptions of the respondents on the correctness of the code with syntax and logical errors?
    b. assessment of the respondents on a code with syntax and logical errors?
    c. perceptions of the respondents on the fairness on the assessment of the teachers on a code with syntax and logical errors?
    d. bases of the respondents in assessing a code with syntax and logical errors?

2. Hypotheses

The following hypotheses were tested using the appropriate statistical tools.

H_{0a}: There is no significant difference between the perceptions of the respondents on the correctness of the code with syntax and logical errors.

H_{0b}: There is no significant difference between the assessment of the respondents on a code with syntax and logical errors.

H_{0c}: There is no significant difference between the perceptions of the respondents on the fairness on the assessment of the teachers on a code with syntax and logical errors.

H_{0d}: There is no significant difference between the bases of the respondents in assessing a code with syntax and logical errors.

3. Methodology

3.1 Research Design, Research Locale, and Subjects

The study is a descriptive design. Second year students of the College of Computer Studies and Systems of the University of the East were selected as the respondents of the study. They were enrolled in Computer Programming Course in Java Programming 2 (PROG2).

3.2 Population and Sample Size

There were 431 students enrolled in PROG2. Eighty-three (83) students who participated in the pretest of the questionnaire were not included in determining the actual population. Thus, the actual population considered was 348. A sample size of 186 was computed using Sloven's formula (e=0.05).

3.3 Research Instrument and Data-Gathering Procedure

Students were randomly selected through their class sections. To accommodate low return rate, two hundred sixty-eight (268) forms were distributed. Two hundred fifty-one (251) forms were retrieved and these were all used in the analysis.

The questionnaire included two sections. The first section was composed of a code with a syntax error while the second section consisted of a code with a logical error. The details of the given codes are shown in Table 1.

Two codes with errors (syntax and logical) were presented to the students (i.e., novice programmers). The causes of the errors were already presented since they might not found the errors in the codes.

Perceptions of the respondents on the correctness of the codes (rated from 1 – Totally incorrect to 5 – Almost totally correct), their assessment on the codes (rated from 70-74% (Failed) to 75-100% (Passed)), and their corresponding basis on such assessment were gathered on both sections. To determine the basis of the assessment, seven questions were constructed on each set of code (See Table 2.). They could answer 1 (Disagree) to 5 (Strongly agree) to determine their basis of assessing of the given code. All were found to be valid (factor loading of at least 0.50) and reliable (above the minimum criterion of 0.70) (George and Mallery, 2009, Pallant, 2001). (See Table 2.). The questionnaire can be found in Appendix 1.

Their perceptions on the fairness of teachers' assessment were also determined through the questionnaire. They could answer from 1 (Disagree) to 5 (Strongly agree). Teachers' assessments on Code 1 and Code 2 were determined through informal interviews with three (3) PROG2 teachers. Code 1 was assessed 74 while Code 2 was assessed 72. The scale, mean range, and verbal interpretation are shown in Table 3. The statistical tools used in the treatment of data were mean and paired *t*-test. Mean was utilized to determine the perceived correctness, assessment, bases of assessment, and fairness of teachers' assessment on a given code. Paired t-test was used to determine the significance of the difference on the perceived correctness, assessment, bases of assessment, and fairness of assessment on a code with syntax and logical errors. A 5% level of probability with 95% reliability was adopted to determine the degree of significance of the findings.

4. Results and Discussion

4.1 Code Correctness and Self-Assessment, Perception of Fairness, and Basis of Code Self-Assessment

Most of the respondents were male (f = 187, 74%). Information Technology (f = 239, 95%), Computer Science (f = 8, 3%), and Associate in Technology (f = 4, 2%) students were the respondents of the study. Their perceptions on program correctness, their code assessment, and on the fairness of teachers' assessment are shown in Table 4.

As shown in Table 4, respondents perceived that both codes were "somewhat correct" (Syntax error = 3.22; Logical error = 3.06). This was found to be consistent with the studies of Ben-David Kolikant (2005), and Ben-David Kolikant and Mussai (2008) that even if a code or program had an error, novice programmers believed that the program had still a grain of correctness. It was also revealed that the respondents gave a passing mark for the codes even if the codes had flaws (Syntax error = 82.35, "Passed"; Logical Error = 81.29, "Passed"). Interestingly, they only agreed to a lesser extent (Syntax error = 2.60, "Moderately agree"; Logical error = 2.54, "Moderately agree") on the assessment of the teachers on the flawed codes.

The study attempted to explain why such misconceptions existed. The bases of the respondents on assessing the codes are shown in Table 5. On a code with syntax error, the reason "*the error can be fixed easily*" got the highest mean rating of 3.99 ("Agree"). This implies that, in the context of this study, a passing rate would be given to a code with syntax error since they believed that the error could be corrected easily. As such, they would moderately agree that the rating of the teachers on Code 1 as 74 ("Failed") was fair. On the other hand, the top reason of giving a passing rate of 81.29 ("Passed") to a code with logical error was attributed on the basis that "*it was fair to give points for effort*" (mean rating = 3.67, "Agree").

It is also apparent that novice programmers have two sets of rating parameters depending on the nature of the error. Codes with syntax error were graded based on the "difficulty" of debugging the code. On the other hand, codes with logical error were rated based on the efforts exerted in doing the codes. These findings have an implication on assessing the programming exercises of the students. While teachers do not nurture or tolerate students' misconceptions on program correctness by giving a non-passing mark, students, on the other hand, perceived that the codes should have been given a passing mark. The findings suggest that teachers should clearly explain the rating parameters on assessing a code on the first day of class and it should be written explicitly on the course syllabus. In this manner, teacher-student conflict can be avoided.

4.2 Test of Difference between the Perceived Correctness, Self-Assessment, Perceived Fairness of Teachers' Assessment, and Bases of Assessment of the Respondents on a Code with Syntax and Logical Errors

As shown in Table 4, the respondents have higher mean ratings on the correctness of code with syntax error than on a code with logical error. However, the difference between the mean rating on the perceived correctness of the two codes (D = 0.17) was not found to be significant ($t(250) = 1.760$, p-value > 0.05) as can be seen in Table 6.

Also, the difference on the perception of fairness in assessing a code with syntax and logical errors was not found to be significant (D = 0.06, $t(250) = 0.653$, p-value > 0.05). In other words, respondents perceived that the rating of the teachers on both codes were unfair. The implications of these findings are twofold. First, the previous findings of Ben-David Kolikant (2005), and Ben-David Kolikant and Mussai (2008) that novice programmers had misconceptions of program correctness were further explained. This study reveals that program correctness misconceptions can be manifested *across* syntax and logical errors. In other words, novice programmers would perceive a flawed code to be somewhat correct regardless of its error. Second, students would perceive that the assessment of the teachers (i.e., a failing mark) was unfair regardless of the types of error.

Meanwhile, respondents rated a code with syntax error higher than a code with logical error (See Table 4.). As shown in Table 6, this difference is found to be significant and it is unlikely to have arisen from sampling error (D = 1.06, $t(250) = 2.570$, p-value < 0.05). It is revealed that the respondents gave not only a passing rate to a code with syntax error but also a higher passing mark to a code with syntax error. Thus, it appeared that novice programmers had not only misconceptions of program correctness but also wrong notions of code assessment and fairness in code assessment.

Furthermore, Table 7 shows the test of difference between the reasons of the respondents on the assessment on a code with syntax and logical errors. In terms of Reason 1, respondents have higher mean ratings (D = 0.219) on a code with syntax error than on a code with logical error. The $t(250)$-value of 2.396 with an associated p-value < 0.05 shows that the result of the difference is unlikely to have arisen from sampling error. This implies that the respondents perceived syntax error as a *lesser* serious error than a logical error.

Meanwhile, in Reason 2, D = -0.259. This shows that respondents tend to *tolerate* a logical error more than a syntax error. The result is unlikely to have arisen from sampling error ($t(250) = -3.259$, p-value < 0.05). They also perceived that syntax error is easier to *fix* (Reason 3, D = 0.339, t-value = 4.711, $df = 250$, p-value < 0.05) and *detect* (Reason 4, D = 0.323, t-value = 4.153, $df = 250$, p-value < 0.05) than logical error. Meanwhile, students have higher mean rating on giving more points for *effort* (Reason 6, D = 0.151, t-value = 2.447, $df = 250$, p-value < 0.05) and *credits* (Reason 7, D = 0.259, t-value = 3.507, $df = 250$, p-value < 0.05) on a syntax error than a logical error.

The findings previously presented give new perspectives on how novice programmers rate codes. First, novice programmers measure the gravity of the error on the type of error the code has and in return, they use it as basis to rate a code. Second, a higher rating can be given if the flaw in the code is easier to detect and to fix.

Third, it is not new that novice programmers would tolerate errors. However, in this study, it was found out that they would tolerate a logical error more than a syntax error. This can be attributed to two reasons. The first reason is attributed to the previously reported findings of Ben-David Kolikant and Mussai (2008) that novice programmers would consider a code correct that would work in general but not entirely correct for the whole valid inputs (See Table 1.). This implies that students are "output-based raters", i.e., they would rate a code with logical error because *at the very least* it could produce an output.

Second, the novice programmers' duality of rating a code was applied. As shown in Table 4 and Table 7, students perceived that logical error was more difficult to detect and to fix than a syntax error. As such, a passing rate on logically incorrect code was justified through the efforts they exerted on doing the code.

These new findings have implications in the field of computing education. The findings would be beneficial to computing educators to understand better novice programmers (i.e., students). This extends the findings of Ben-David Kolikant (2005) that novice programmers had alternative standards of program correctness.

This study shows that novice programmers also have alternative standards of grading the codes. Educators should emphasize that an error is an error regardless of its nature and programming exercises of the students should be graded accordingly. On the other hand, educators could devise a grading system that would be fair at the level of the students. This would be difficult on the part of teachers since students would expect that their teachers would apply the duality of code assessments; therefore, a passing mark would be expected. At any rate, teachers should rate the programming exercises in a fair manner without nurturing the wrong notions of program correctness, code assessment, and fairness in code assessment.

5. Conclusions and Recommendations

On the basis of the findings presented previously, the following conclusions are drawn.

- The null hypothesis stating that (H_{0a}) there is no significant difference between the perceptions of the respondents on the correctness of the code with syntax and logic errors is accepted.
- The null hypothesis stating (H_{0b}) that there is no significant difference between the assessment of the respondents on a code with syntax and logical errors is rejected.
- The null hypothesis stating that (H_{0c}) that there is no significant difference between the perceptions of the respondents on the fairness on the assessment of the teachers on a code with syntax and logical errors is accepted.
- The null hypothesis stating that (H_{0d}) that there is no significant difference between the bases of the respondents in assessing a code with syntax and logical errors is partially rejected.

These conclusions extended the findings of the existing literature. The study presented new findings about novice programmers. They (1) tend to find a grain of correctness on codes regardless of the nature of the errors of the code, (2) gave a higher mark on a code with syntax error than a code with logical error, (3) perceived that the teachers' assessment on codes was unfair regardless the nature of the error of the codes, and (4) rated a code with syntax error higher than a code with logical error because they perceived that syntax error was insignificant and easy to fix and detect. They also based their rating on giving credits on the efforts exerted in doing the code. The study also found out that novice programmers had the misconceptions of code assessment and fairness of code assessment. As such, computing educators were given new insights on understanding novice programmers.

The findings could serve as basis for computing educators in teaching novice programmers. They have a big role in educating the students in correcting these new misconceptions. A clear grading policy set forth at the start of the semester should be given to avoid conflict. Nevertheless, teachers should balance their grading system on the grounds of fairness and quality. The grading system should be lenient but not too lax to avoid nurturing the wrong conceptions of program correctness, code assessment, and fairness in assessment. A follow-up study on the effectiveness of this approach is, therefore, highly recommended.

6. Acknowledgments

This piece of work is made possible because of the valuable support of Dr. Ester A. Garcia, Dr. Olivia C. Caoili, Dean Rodany A. Merida, and Dr. Socorro R. Villamejor.

References

Ben-David Kolikant, Y. (2005). Students' alternative standards for correctness. *Proceedings of the 2005 International workshop on computing education research* (pp. 37–43). New York: ACM Press.

Ben-David Kolikant, Y., & Mussai, M. (2008). "So my program doesn't run!" Definition, origins, and practical expressions of students' (mis)conceptions of correctness. *Computer Science Education, 18*(2), 135–151.

Carbone, A., Hurst, J., Mitchell, I., & Gunstone, D. (2009). An exploration of internal factors influencing student learning of programming. In Md. J. Nordin, K. Jumari, M. S. Zakaria, & Suwarno (Eds.). *Computing Education 2009: Proceedings 11th Australasian Computing Education Conference (ACE 2009), Conferences in Research and Practice in Information Technology (CRPIT), 95* (pp. 25–34). Sydney, Australia: Australian Computer Society, Inc.

Ebrahimi, A. (1994). Novice programmer errors: Language constructs and plan composition. *International Journal of Human-Computer Studies, 41*, 457–480. doi:10.1006/ijhc.1994.1069.

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: Finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education, 18*(2), 93–116.

George, D., & Mallery, P. (2009). SPSS for Windows step by step: A simple guide and reference 16.0 update 9th edn. Boston: Pearson Education.

Gobil, A. R. M., Shukor, Z., & Mohtar, I. A. (2009). Novice difficulties in selection structure. In Md. J. Nordin, K. Jumari, M. S. Zakaria, & Suwarno (Eds.). *2009 International Conference on Electrical Engineering and Informatics, 2*, (pp. 351–356). doi: 10.1109/ICEEI.2009.5254715. New Jersey, USA: IEEE Computer Society.

Jackson, J., Cobb, M., & Carver, C. (2005). Identifying top java errors for novice programmers. *Frontiers in Education Conference*, *1*, T4C-T4C27. doi: 10.1109/FIE.2005.1611967.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, *37*(2), 83–137.

Lenox, T. L., Woratschek, C. R., & Davis, G. A., (2008). Exploring declining CS/IS/IT enrollments. *Information Systems Education Journal*, *6*(44), 1–11.

Pallant, J. (2001). *SPSS survival manual: A step by step guide to data analysis using SPSS for Windows version 10*. Open University Press, Buckingham.

Pendergast, M. O. (2006). Teaching introductory programming to IS students: Java problems and pitfalls. *Journal of Information and Technology Education*, *5*, 491–595. Retrieved January 8, 2011, from http://jite.org/documents/Vol5/v5p491-515Pendergast128.pdf

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172. doi:10.1076/csed.13.2.137.14200.

Sloan, R. H., & Troy, P. (2008). CS 0.5: A better approach to Introductory Computer Science for Majors. *ACM SIGCSE Bulletin*, *40*(1), 271–275. doi: 10.1145/1352322.1352230.

Table 1. Program Codes with Error(s)

CODE 1 – SYNTAX ERROR		**CAUSE OF ERROR**
1	int number = scanner.nextInt();	
2		
3	if (number < 0) {	
4	System.out.println(number + "is NEGATIVE.")	Line 4: no semi-colon
5	} else if (number > 0) {	
6	System.out.println(number + "is POSITIVE.");	
7	} else {	
8	System.out.println("Number is ZERO.");	
9	}	
CODE 2 – LOGICAL ERROR		**CAUSE OF ERROR**
1	BufferedReader br = new BufferedReader(new	
2	InputStreamReader(System.in));	Line 5: the symbol “>” will exclude 75 as a passing grade.
3	int grade;	
4	System.out.print("Input a number: ");	
5	grade = Integer.parseInt(br.readLine());	
6	if(grade>75){	
7	System.out.println("PASSED");	
8	}	
9	else{	
10	System.out.println("FAILED");	
	}	

Table 2. Validity and Reliability of the Questions

Reasons	**Syntax Error**		**Logical Error**	
	Factor loading	**Cronbach's alpha**	**Factor loading**	**Cronbach's alpha**
1. The error is just a minor one.	0.677	0.707	0.780	0.803
2. The error is tolerable.	0.502		0.699	
3. The error can be fixed easily.	0.741		0.852	
4. The error can be detected easily.	0.554		0.810	
5. The code has just one error.	0.638		0.510	
6. It is fair to give points for effort.	0.552		0.573	
7. I give credit to its correct logic/syntax.	0.739		0.519	

Table 3. The 5-Point Scale, Its Mean Range and Verbal Interpretation

Scale	Mean Range	Verbal Interpretation
5	4.51 – 5.00	Almost totally correct / Strongly agree
4	3.51 – 4.50	Partially correct / Agree
3	2.51 – 3.50	Somewhat correct / Moderately agree
2	1.51 – 2.50	Incorrect / Slightly agree
1	1.00 – 1.50	Totally incorrect / Disagree

Table 4. Perceptions of the Respondents in terms of Program Correctness, Code Assessment, and Fairness of Assessment

Perception	Mean	Verbal Interpretation
Correctness of Code with		
Syntax error	3.22	Somewhat correct
Logical error	3.06	Somewhat correct
Assessment of Code with		
Syntax error	82.35	Passed
Logical error	81.29	Passed
Fairness in Assessing a Code with		
Syntax error	2.60	Moderately agree
Logical error	2.54	Moderately agree

Table 5. Bases of the Respondents in Assessing a Code with Syntax and Logical Errors

Reasons	Syntax Error		Logical Error	
	Mean	**V.I.**	**Mean**	**V.I.**
1. The error is just a minor one.	3.31	Moderately agree	3.10	Moderately agree
2. The error is tolerable.	2.62	Moderately agree	2.88	Moderately agree
3. The error can be fixed easily.	**3.99**	**Agree**	3.65	Agree
4. The error can be detected easily.	3.85	Agree	3.53	Agree
5. The code has just one error.	3.59	Agree	3.54	Agree
6. It is fair to give points for effort.	3.82	Agree	**3.67**	**Agree**
7. I give credit to its correct logic/syntax.	3.75	Agree	3.49	Moderately agree

Table 6. Test of Difference between the Perceptions of Novice Programmers on a Code with Syntax Error and Logical Error

Perception of Novice Programmers on	Mean Difference (D) (Syntax – Logical)	*t*-value[a]	Sig.
Correctness of Code	0.17	1.760	0.080
Assessment on the Code	1.06	2.570	0.011
Fairness of Assessing a Code	0.06	0.653	0.514

[a] $df = 250$

Table 7. Test of Difference between the Reasons of Code Assessment of the Respondents on a Code with Syntax and Logical Errors

Reasons	Mean Difference (D) (Syntax – Logical)	*t*-value[a]	Sig.
1. The error is just a minor one.	0.219	2.396	.017
2. The error is tolerable.	-0.259	-3.259	.001
3. The error can be fixed easily.	0.339	4.711	.000
4. The error can be detected easily.	0.323	4.153	.000
5. The code has just one error.	0.056	0.681	.497
6. It is fair to give points for effort.	0.151	2.447	.015
7. I give credit to its correct logic/syntax.	0.259	3.507	.001

[a] $df = 250$

Appendix 1

PART I

Suppose the code below is a fragment of a program. The code below determines whether a number is positive or negative. It has **ONE** error. See the table below for details and answer the questions associated with the code.

	CODE 1	CAUSE OF ERROR
1	int number = scanner.nextInt();	
2		
3	if (number < 0) {	
4	System.out.println(number + "is NEGATIVE.")	Line 4: no semi-colon
5	} else if (number > 0) {	
6	System.out.println(number + "is POSITIVE.");	
7	} else {	
8	System.out.println("Number is ZERO.");	
9	}	

1.1 How do you perceive the correctness of the code? Please check (✓)

[ ] Totally incorrect [ ] Somewhat correct [ ] Partially correct
[ ] Incorrect [ ] Almost correct

1.2 Using the grades from 70% to 74% (failed) and 75% to 100% (passed), grade the code above.

Put your grade here: _____ %

1.3 Using the scale below, answer the following questions. Please check (✓)

1 – Disagree 3 – Moderately agree 4 – Agree
2 – Slightly agree 5 – Strongly agree

I gave such grade because	1	2	3	4	5
1. The error is just a minor one.	[ ]	[ ]	[ ]	[ ]	[ ]
2. The error is tolerable.	[ ]	[ ]	[ ]	[ ]	[ ]
3. The error can be fixed easily.	[ ]	[ ]	[ ]	[ ]	[ ]
4. The error can be detected easily.	[ ]	[ ]	[ ]	[ ]	[ ]
5. The code has just one error.	[ ]	[ ]	[ ]	[ ]	[ ]
6. It is fair to give points for effort.	[ ]	[ ]	[ ]	[ ]	[ ]
7. I give credit to its correct logic/syntax.	[ ]	[ ]	[ ]	[ ]	[ ]

1.4 A professor gave a 74% grade on the given code. The professor believes that 74% grade is a fair grade for this code. Do you agree with the professor? Indicate your response below by checking (✓) the box.

[ ] Disagree [ ] Slightly agree [ ] Moderately agree [ ] Agree [ ] Strongly agree

PART II

Suppose the code below is a fragment of a program. The code below accepts a grade from the user. Then, it determines whether the grade passed or failed. A grade less than 75 is failed; otherwise, passed. It has **ONE** error. See the table below for details and answer the questions associated with the code.

	CODE 2	**CAUSE OF ERROR**
1	BufferedReader br = new BufferedReader(new InputStreamReader(System.in));	
2	int grade;	
3	System.out.print("Input a number: ");	
4	grade = Integer.parseInt(br.readLine());	
5	if(grade>75){	Line 5: the symbol ">" will exclude 75 as a passing grade.
6	System.out.println("PASSED");	
7	}	
8	else{	
9	System.out.println("FAILED");	
10	}	

2.1 How do you perceive the correctness of the code? Please check (✓)

[ ] Totally incorrect [ ] Somewhat correct [ ] Partially correct
[ ] Incorrect [ ] Almost correct

2.2 Using the grades from 70% to 74% (failed) and 75% to 100% (passed), grade the code above.

Put your grade here: _____ %

2.3 Using the scale below, answer the following questions. Please check (✓)

1 – Disagree 3 – Moderately agree 4 – Agree
2 – Slightly agree 5 – Strongly agree

I gave such grade because	**1**	**2**	**3**	**4**	**5**
1. The error is just a minor one.	[ ]	[ ]	[ ]	[ ]	[ ]
2. The error is tolerable.	[ ]	[ ]	[ ]	[ ]	[ ]
3. The error can be fixed easily.	[ ]	[ ]	[ ]	[ ]	[ ]
4. The error can be detected easily.	[ ]	[ ]	[ ]	[ ]	[ ]
5. The code has just one error.	[ ]	[ ]	[ ]	[ ]	[ ]
6. It is fair to give points for effort.	[ ]	[ ]	[ ]	[ ]	[ ]
7. I give credit to its correct logic/syntax.	[ ]	[ ]	[ ]	[ ]	[ ]

2.4 A professor gave a 72% grade on the given code. The professor believes that 72% grade is a fair grade for this code. Do you agree with the professor? Indicate your response below by checking (✓) the box.

[ ] Disagree [ ] Slightly agree [ ] Moderately agree [ ] Agree [ ] Strongly agree