Education-oriented Java Compiler with e-TA

Shun HATTORI and Hiroyuki KAMEDA

School of Computer Science, Tokyo University of Technology
1404–1 Katakura, Hachioji, Tokyo 192–0982, Japan
hattori@cs.teu.ac.jp

Abstract

In recent years, those who do not acquire enough programming ability and also hesitate in programming have been increasing, even though they are Computer Science students/graduates. One of the major reasons is that they could not avoid facing very difficult and unexpectedly massive compile errors with unknown technical words when they were beginners of programming. This paper proposes an education-oriented compiler with e-TA (electronic Teaching Agent) for programming education, and shows an implementation of our education-oriented Java compiler to present not only static compile errors as-is but also easy-to-understand advice appropriately and on a timely basis (dynamically) depending on programming knowledge of learners who browse the compile errors as if at least a high-level human teacher or TA were always man-on-man standing at the learners' side.

Keywords: Education-Oriented Compiler, e-TA (electronic Teaching Agent), User-Adaptive Compiler, Proactive Compiler, Programming Education.

1 Introduction

Today, software is one of the most important social infrastructures. Our societies require capable human resources who can develop software at the highest level, and various educational institutions such as universities provide software engineering (programming) education aggressively. In Japan, however, those who do not acquire enough programming ability and also hesitate in programming have been increasing in recent years, even though they are CS (Computer Science) or IT (Information Technology) students/graduates.

One of the major reasons is that they could not avoid facing very difficult and unexpectedly massive compile errors with unacquainted English words or technical words and thus have given up learning about programming knowledge and skills when they were beginners of programming. Conventional compilers such as javac [2] are for not programming education but software development, and are for not beginners of programming but intermediate and advanced developers. Another reason is that such human-powered supports by a teacher and a few TAs (Teaching Assistants) as deciphering compiler's error messages for learners and giving appropriate advice timely have several limitations in introductory programming exercises [1].

In this paper, we propose an education-oriented compiler with user-adaptive and active e-TA (electronic Teaching Agent) for introductory programming exercises, and describe an implementation of our education-oriented Java compiler to present not only static and often difficult-to-understand error messages by the most primary Java compiler, javac, but also easy-to-understand advice appropriately and on a timely basis (dynamically) depending on programming knowledge of learners who browse the compiler's error messages as if a high-level teacher or TA were always man-on-man standing at the learners' side.

Of course, there are other approaches except our approach of using our proposed education-oriented Java compiler on the CLI (Command Line Interface) as follows: using the other Java compiler as Jikes [3], GNU Compiler for Java (GCJ) [4], and Eclipse Compiler for Java (ECJ) [5]; using such a GUI integrated development environment (IDE) as Ecilpse and NetBeans [6] with keyword auto-complete (suggest) and high-level debug functions etc.; using such a programming language for new learners of programming as Nigari [7, 8, 9] and Kotodama [10, 11, 12]; using such a programming (software engineering) environment for novices as PEN [13, 14, 15] and Robotran [16].

However, the educational objective of our programming exercises for novices at our university is not only to make them capable of editing correct Java source code all by themselves, but also to let them practice away at typing, get used to the CLI, logically modify their source code with the help of compiler's error messages, and we cannot easily change our assumed programming process for students, which consist of editing their source code by such an editor as Emacs, compiling their source code by the most conventional Java compiler, javac, on the CLI, and modifying and recompiling their source code if any error. And javac must be the most primary Java compiler, and there are not a few Java development environments such as Greenfoot [17] which adopt javac as an internal Java compiler. Therefore, our proposed education-oriented Java compiler tries to generate advice for javac's error messages as appropriately and timely as possible.

2 Problems in Introductory Java Programming Exercises

This section discusses problems of using the most conventional Java compiler, javac [2], in introductory Java programming education, and extracts the requirements of our proposed education-oriented compiler with e-TA (electronic Teaching Agent).

The below "HJW.java (model answer)" is one of the most typical Java introductory source codes. At our university, Computer Science (CS) freshmen are first instructed to edit it without modification by such an editor as Emacs, and to compile it by the most primary Java compiler, javac, on the CLI.

HJW.java (model answer)

```
class HJW {
  public static void main(String[] args) {
    System.out.println("Hello Java World");
  }
}
```

For such an assignment to type the model answer without thinking and modification, not a few CS students make mistakes. For example, a new learner of Java programming might make a typo in the 1st line, the reserved keyword "`class`" to "`clas`", as shown in "HJW.java (typo)". In fact, we often encounter many typos of English reserved keywords in our programming exercises, because many Japanese students are not good at English and dislike English words.

HJW.java (typo)

```
clas HJW {
  public static void main(String[] args) {
    System.out.println("Hello Java World");
  }
}
```

Of course, the new learner himself intended to type the model answer without modification, and would make no question of success of compiling his typed source code. However, the primary Java compiler, javac (in JDK1.6), offers the following 3 error messages for his source code with a typo, and thus he would be surprised at his unexpected error messages and be at odds as to how to modify them.

javac HJW.java (typo)

```
HJW.java:1: class, interface, or enum expected
clas HJW {
^
HJW.java:2: class, interface, or enum expected
  public static void main(String[] args) {
                     ^
HJW.java:4: class, interface, or enum expected
  }
  ^
3 errors
```

The above error messages by the conventional Java compiler, javac, seem to have the following problems. Novices for Java programming unexpectedly encounter such high-level (not entry-level) reserved keywords as "`interface`" and "`enum`" in every error message even while tackling such an entry-level assignment to type the most introductory Java source code without modification. They must have not yet learned such high-level reserved keywords. Facing unknown and difficult concepts could be huge barriers (stress) for new learners. Therefore, our proposed education-oriented Java compiler needs to mask unknown technical keywords in its error messages depending on knowledge of learners who browse the compile errors, or to supplement them with some additional explanation and/or by referring textbooks and/or Web pages.

And the multiple error messages do not show which error message to be first tackled and how to make a tangible modification. Rather, offering only the 1st error message is informative. The other error messages are not helpful but harmful, because there is a typo in the 1st line of his source code and the 2nd or 3rd error message points that there are some error in the 2nd or 4th line respectively. Offering such harmful error messages that point the location in which there is no error and offering unexpectedly many error messages (much more than the number of errors embedded actually) is very unfriendly for new learners. Therefore, our proposed education-oriented Java compiler needs to mask such harmful error messages and to reduce its error messages to the number of errors embedded actually.

Another error for the most introductory Java source code is forgetting to close a block by "`}`" as shown in "HJW.java (missing of `}`)". Not just new learners but any programmers often make correspondence mistakes of "`{`" and "`}`", and it is also not easy for teachers and TAs to modify them quickly. In addition, most new learners often do not indent properly.

HJW.java (missing of })

```
class HJW {
public static void main(String[] args) {
System.out.println("Hello Java World");
}
_
```

The most primary Java compiler, javac (in JDK1.6), offers the following error message for the above source code "HJW.java (missing of `}`)".

javac HJW.java (missing of })

```
HJW.java:4: reached end of file while parsing
}
 ^
1 error
```

This error message, especially a technical word "parsing" is cryptic and not helpful for new learners. In this case, a human teacher or TA had better give "Indent properly." as more educational advice before giving "Insert `}` in the 5th line." as the last and most specific advice to students.

The above-mentioned problem of the primary Java compiler is difficulty for users, especially new learners, to understand its compile error messages, i.e., "disability of user-adaptiveness". Another problem of the primary Java compiler is passivity for users, especially some learners who cannot ask a human teacher or TA because of such a personality as shyness of strangers even if stuck in compiler's error messages, i.e., "disability of activeness". Therefore, our proposed education-oriented Java compiler with e-TA should have both "user-adaptiveness" and "activeness".

3 Design of Education-oriented Java Compiler with e-TA

This section describes a design of our proposed education-oriented Java compiler, edujavac, in programming education for novices, as a 1st step to the universal compiler whose error messages are easy-to-understand for anybody with any level of knowledge and skills of programming. Not to overstress new learners of programming by difficult-to-understand compiler's error messages and/or human-to-human communication such as question-and-answering, not a human teacher or TA but a computer e-TA (electronic Teaching Agent) of our proposed education-oriented compiler provides the learners with such supports as deciphering compiler's error messages for learners and giving appropriate advice about what wrong in their source code and/or how to modify their source code.

3.1 Requirements

Our proposed education-oriented Java compiler with e-TA should have the following requirements.

1. User-Adaptiveness
    - **Learner-Adaptiveness**: In programming exercises, a human teacher or TA seems to offer a targeted student with appropriate and timely advice based on the targeted student's level of knowledge and skills of programming inferred by observing the targeted student's action history (e.g., transition of source code) and error tendency from behind, and/or asking the targeted student additional questions arbitrarily (if necessary, referring the example source code of the targeted student's tackling assignment. Therefore, for a learner, our proposed e-TA should generate appropriate and timely advice dependent on the learner's level of knowledge (e.g., reserved keywords and technical terms) of programming.
    - **Teacher-Adaptiveness**: Before programming exercises, a human teacher had better give his TAs enough guidance about how to advise his students, e.g., timing control and kindness control. Therefore, for a teacher, our proposed e-TA should generate appropriate and timely advice according to the teacher's educational policies about timing, kindness, and programming knowledge and skills to teach.
2. **Activeness**: Some learners who cannot ask a human teacher or TA because of such a personality as shyness of strangers even if stuck in compiler's error messages and would leave them unsolved. Therefore, e-TA should not reactively but proactively advise shy learners if necessary without their explicit search of the other's advice.

3.2 Basic Architecture

Figure 1 gives an overview of our proposed education-oriented compiler with e-TA who generates appropriate and timely advice to error messages by compiling a learner's source code for a teacher's assignment based on Learner and Teacher models.

Each learner has a knowledge directory, know-dir, on his local computer as her Learner model. Technical keywords with high frequency to some extent extracted from electronic documents in the learner's knowledge directory are identified as his well-known keywords. Unknown technical keywords in its error messages are masked depending on the user's knowledge (i.e., electronic documents in the knowledge directory), or are supplemented with some additional explanation and by referring textbooks and Web pages. Whether to mask or supplement them for each learner and what to refer are specified in advance and can be always switched by each teacher as his Teacher model.

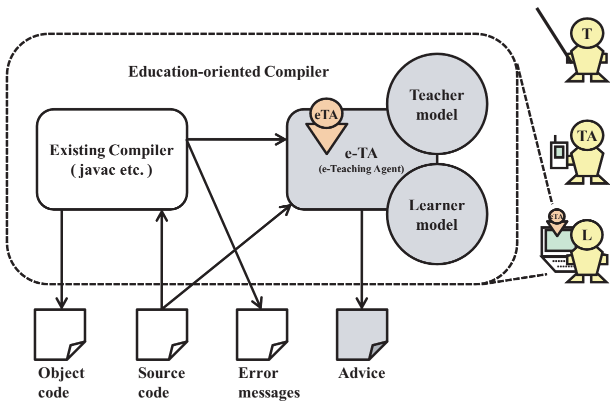

Figure 1: Education-oriented Compiler with e-TA.

3.3 Advice Generation

First, our e-TA finds the most effective (but maybe the least educational) method to modify a targeted source code with errors by referring its original error messages of the most conventional Java compiler, javac.

In the case of generating the most effective modification method to the javac's 1st error message for "HJW.java (typo)" in Section 2, e-TA collects the following 6 kinds of modification candidates by inferring "insert" or "replace" as its operation from a pattern "expected" and extracting its keyword and location, and selects one based on the number of re-compile errors after applying each modification candidate:

(insert, 1, 1, "class")	$3 \to 1$ error
(insert, 1, 1, "interface")	$3 \to 1$ error
(insert, 1, 1, "enum")	$3 \to 2$ errors
(replace, 1, 1, "class")	$3 \to \mathbf{0}$ **error**
(replace, 1, 1, "interface")	$3 \to 2$ errors
(replace, 1, 1, "enum")	$3 \to 4$ errors

Next, our e-TA generates a sequence of advices by rule-based interpolating between the javac's error message(s) and the most effective modification method.

4 Demos of Education-oriented Java Compiler with e-TA

As an implementation of our proposed education-oriented Java compiler, edujavac, in programming education, this section describes a prototype of knowledge-dependent Java compiler with reactive or proactive e-TA to present not only static error messages by the existing Java compiler, javac, but also easy-to-understand advice appropriately and on a timely basis (dynamically) depending on programming knowledge of learners who browse the compiler's error messages. The current prototype can deal with a typo, missing of "}" or ")", and not-found symbol as a part of various kinds of compile errors.

4.1 Advice Examples of Reactive e-TA

The primary Java compiler, javac, offers the quite same error messages indefinitely unless the inputted source code is modified as shown in Section 2. Meanwhile, our proposed education-oriented Java compiler, edujavac, with reactive e-TA offers the following sequence of advices about the source code with a typo, "HJW.java (typo)", to the learner with known technical keyword(s), know-dir={"class"}.

edujavac HJW.java (typo) with Reactive e-TA
[**mask** except know-dir={"class"}]

```
> edujavac HJW.java
HJW.java:1: class expected
clas HJW {
^
HJW.java:2: class expected
  public static void main(String[] args) {
                     ^
HJW.java:4: class expected
  }
  ^
3 errors

> edujavac HJW.java
HJW.java:1: class expected
clas HJW {
^
> edujavac HJW.java
HJW.java:1: class expected
clas HJW {
^
> edujavac HJW.java
HJW.java:1: there is a typo anywhere

> edujavac HJW.java
HJW.java:1: there is a typo in the line no.1

> edujavac HJW.java
HJW.java:1: replace clas with something in the...
 line no.1

> edujavac HJW.java
HJW.java:1: replace clas with class in the line no.1
```

For the 1st/2nd edujavac command by the learner, e-TA passively offers the error message(s) in which his unknown technical keywords such as "interface" and "enum" are masked not to overstress the learner by difficult-to-understand error messages. For the 3rd edujavac command, e-TA passively offers the same error message as for the 2nd one because the interval between the 2nd and 3rd edujavac commands is shorter than the teacher's specified interval of updating content of e-TA's advices.

Our edujavac with e-TA masks the other technical keywords than a learner's know-dir = {"class"} as above, while it supplements them with their explanations as his teacher's knowledge to teach as below.

edujavac HJW.java (typo) with Reactive e-TA
[**supplement** except know-dir={"class"}]

```
> edujavac HJW.java
HJW.java:1: class, interface or enum expected
clas HJW {
^
HJW.java:2: class, interface or enum expected
  public static void main(String[] args) {
                     ^
HJW.java:4: class, interface or enum expected
  }
  ^
3 errors

interface is an abstract type that ...
  TEXT: Objects First with Java, pp.328
   URL: http://en.wikipedia.org/wiki/Interface_(Java)

enum is a data type that ...
  TEXT: Objects First with Java, pp.233
   URL: http://en.wikipedia.org/wiki/Enumerated_type

> edujavac HJW.java
HJW.java:1: class, interface or enum expected
clas HJW {
^
interface is an abstract type that ...
  TEXT: Objects First with Java, pp.328
   URL: http://en.wikipedia.org/wiki/Interface_(Java)

enum is a data type that ...
  TEXT: Objects First with Java, pp.233
   URL: http://en.wikipedia.org/wiki/Enumerated_type

> edujavac HJW.java
HJW.java:1: class expected
clas HJW {
^
> edujavac HJW.java
HJW.java:?: there is a typo anywhere

> edujavac HJW.java
HJW.java:1: there is a typo in the line no.1

> edujavac HJW.java
HJW.java:1: replace clas with something in the...
 line no.1

> edujavac HJW.java
HJW.java:1: replace clas with class in the line no.1
```

As below, our edujavac with reactive e-TA offers more easy-to-understand and varied advices about the quite same source code with a typo, "HJW.java (typo)", dependent on known technical keywords, know-dir, as knowledge of a learner and policies on how to support, e.g., mask/supplement and timing control, as knowledge of her teacher.

edujavac HJW.java (typo) with Reactive e-TA
[**mask** except know-dir={"class", "interface"}]

```
> edujavac HJW.java
HJW.java:1: class or interface expected
clas HJW {
^
HJW.java:2: class or interface expected
  public static void main(String[] args) {
                  ^
HJW.java:4: class or interface expected
  }
  ^
3 errors

> edujavac HJW.java
HJW.java:1: class or interface expected
clas HJW {
^

> edujavac HJW.java
HJW.java:1: class expected
clas HJW {
^
```

As below, our edujavac with reactive e-TA offers the 1st advice about the source code with a missing of "}", "HJW.java (missing of })", by supplementing the javac's original error messages with teacher-specified explanations of a learner's unknown technical word "parsing". Until she indents her source code properly by using such an editor as Emacs, our edujavac with e-TA continues to give the same advice "indent properly" before giving "insert } in the line no.5" as the last and most specific advice to her.

edujavac HJW.java (missing of }) with Reactive e-TA
[**supplement** except know-dir={"class", "interface"}]

```
> edujavac HJW.java
HJW.java:4: reached end of file while parsing
}
 ^
1 error

parsing is the process of analyzing a text ...
  TEXT: none
   URL: http://en.wikipedia.org/wiki/Parsing

> edujavac HJW.java
HJW.java:?: indent properly

indent is to format code to improve readability ...
  TEXT: Objects First with Java, pp.494
   URL: http://en.wikipedia.org/wiki/Indent_style

> edujavac HJW.java
HJW.java:?: indent properly

indent is to format code to improve readability ...
  TEXT: Objects First with Java, pp.494
   URL: http://en.wikipedia.org/wiki/Indent_style

> emacs HJW.java

> more HJW.java
class HJW {
  public static void main(String[] args) {
    System.out.println("Hello Java World");
  }

> edujavac HJW.java
HJW.java:5: insert } in the line no.5
```

4.2 Advice Examples of Proactive e-TA

Our education-oriented Java compiler with proactive e-TA, edujavac, offers the following advices to the learner with known technical keywords, know-dir={"class", "interface"}. For a Java filename (.java) at the 1st learner's prompt (`you>>`), our e-TA reactively offers all of its javac's error messages in which her unknown technical keywords such as "enum" are masked. Even if no input as the 2nd learner's prompt, after such a period of time as 15 minutes specified by her teacher, our e-TA proactively offers only the 1st error message in which her unknown technical keywords are masked. For a technical keyword at the 3rd learner's prompt, our e-TA reactively offers the explanation of the keyword specified by her teacher. Even if no input as her prompt, our e-TA proactively offers one by one from a generated sequence of advices after such a period.

edujavac HJW.java (typo) with Proactive e-TA
[**mask except know-dir={"class", "interface"}**]

```
> edujavac
eTA>> Hello.

you>> HJW.java
HJW.java:1: class or interface expected
clas HJW {
^
HJW.java:2: class or interface expected
  public static void main(String[] args) {
                  ^
HJW.java:4: class or interface expected
  }
  ^
3 errors

you>>

eTA>> HJW.java:1: class or interface expected
clas HJW {
^

you>> interface

eTA>> interface is an abstract type that ...
  TEXT: Objects First with Java, pp.328
   URL: http://en.wikipedia.org/wiki/Interface_(Java)

you>>

eTA>> HJW.java:1: class expected
clas HJW {
^

you>>

eTA>> HJW.java:1: there is a typo anywhere

you>>

eTA>> HJW.java:1: there is a typo in the line no.1

you>>

eTA>> HJW.java:1: replace clas in the line no.1

you>>

eTA>> HJW.java:1: replace clas with class in the...
 line no.1

you>>
```

5 Conclusion

In this paper, we have proposed an education-oriented compiler with user-adaptive and active e-TA (electronic Teaching Agent) about compile errors for not intermediate and advanced program developers but new learners of programming in the context of software engineering education, especially introductory programming exercise. And we have described an implementation of our proposed education-oriented Java compiler to present not only static and often difficult-to-understand error messages by the most primary Java compiler, javac, but also easy-to-understand advice appropriately and on a timely basis (dynamically) depending on programming knowledge of learners who browse the compiler's error messages as if at least a high-level teacher or TA were always man-on-man standing at the learners' side. The prototype can deal with a typo, missing of "}" or ")", and not-found symbol as a part of various kinds of compile errors. We are working up the registered patterns and rules to deal with as many kinds of compile errors as possible.

In the future, we plan to evaluate our proposed education-oriented Java compiler, edujavac, with e-TA who give advices proactively or reactively by masking or supplementing a user's unknown technical keywords, by applying it to practical introductory Java programming exercises for Computer Science freshmen at our university. And we try to invent a mechanism that not single but multiple different e-TAs collaboratively give advices to the user as if a human TA asked the other TA or teacher for help in a practical exercise classroom.

Acknowledgment

This work was supported in part by Tangible Software Engineering Education[1] (Project Leader: Taichi Nakamura, Tokyo University of Technology).

References

[1] Hattori, S., Yamazaki, M., and Kameda, H.: "A Prototype of Education-oriented Java Compiler," Proceedings of the Second Forum on Data Engineering and Information Management (DEIM'10), F8-3 (2010).

[2] Wikipedia - javac: `http://en.wikipedia.org/wiki/Javac` (2010).

[3] Jikes' Home: `http://jikes.sourceforge.net/` (2010).

[4] GCJ: The GNU Compiler for Java: `http://gcc.gnu.org/java/` (2010).

[5] Eclipse: `http://www.eclipse.org/` (2010).

[6] Welcome to NetBeans: `http://www.netbeans.org/` (2010).

[7] Nigari System: `http://www2.eplang.jp/nigari/` (2010).

[8] Cho, S., Kai, M., Kawai, A., Hino, T., Maeshima, S., and Kakehi, K.: "Nigari - A Programming Language and Environment for the First Stage, Leading to Java World," Transactions of Information Processing Society of Japan (IPSJ), Vol.45, No.SIG9 (PRO22), pp.25–46 (2004).

[9] Cho, S., Kakehi, K., Kawai, A., Maeshima, S., and Hino, T.: "Nigari System - Stairway to Java," Proceedings of 2nd IADIS International Conference on e-Society, pp.960–965 (2004).

[10] Kotodama Community: `http://garuda.crew.sfc.keio.ac.jp/kotodamaCommunity/` (2010).

[11] Araki Megumi, Ken Okada, and Hajime Ohiwa. The programming education in elementary school using music box as a theme with using programming system "Kotodama on Squeak". *IPSJ SIG Technical Reports*, Vol.2007, No.12(CE-88), pp.69–75 (2007).

[12] Okada, K., Sugiura, M., Matsuzawa, Y., Araki, M., and Ohiwa, H.: "Programming in Japanese for Literacy Education," IFIP History of Computing and Education 3, pp.171–176, Springer Boston (2008).

[13] PEN (Programming Environment for Novices): `http://grape.media.osaka-cu.ac.jp/PEN/` (2010).

[14] Nishida, T., Harada, A., Nakamura, R., Miyamoto, Y., and Matsuura, T.: "Implementation and Evaluation of PEN: The Programming Environment for Novices," Transactions of Information Processing Society of Japan (IPSJ), Vol.48, No.8, pp.2736–2747 (2007).

[15] Nishida, T., Harada, A., Yoshida, T., Nakamura, R., Nakanishi, M., Toyoda, H., Abe, K., Ishibashi, H., and Matsuura, T.: "PEN: A Programming Environment for Novices – Overview and Practical Lessons –," Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-MEDIA), pp.4755–4883 (2008).

[16] R. Mark Meyer and Debra T. Burhans: "Robotran: A Programming Environment for Novices Using LEGO Mindstorms Robots," Proceedings of the 20th International Florida Artificial Intelligence Research Society (FLAIRS) Conference, pp.321–326 (2007).

[17] Greenfoot - The Java Object World. `http://www.greenfoot.org/` (2010).

[1] `http://www.teu.ac.jp/tangible/e-index.html`