Knowledge-Based Sowtftware Engineering
... (Eds.)
Publisher, 2010

Knowledge-based Compiler with e-TA for Software Engineering Education

Shun HATTORI [a,1] and Hiroyuki KAMEDA [a]
[a] *School of Computer Science, Tokyo University of Technology*

Abstract. Software is an essential infrastructure today. Our societies require capable human resources who can develop softwares at the highest level possible, and various educational institutions such as universities provide software engineering (programming) education aggressively. However, those who do not acquire enough programming ability and also hesitate in programming have been increasing in recent years, even though they are Computer Science (CS) or Information Technology (IT) students/graduates. The main reason is that they could not avoid facing very difficult and unexpectedly massive compile errors with unknown technical words when they were beginners of programming. Another reason is that teachers and TAs (Teaching Assistants) cannot always give timely and appropriate advices to varying knowledge or levels of learners. This paper proposes a knowledge-dependent compiler with e-TA (electronic Teaching Agent) for software engineering education, and shows a prototype 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.

Keywords. Education-oriented Compiler, Knowledge-dependent Compiler, Java Compiler, Software Engineering Education, Programming Education, e-TA (electronic Teaching Agent).

Introduction

Today, software is one of the most important social infrastructures. Software dependency of our societies has been going on increasing year by year. For example, office workers in Japan are no longer able to get anything done without word-processing softwares such as Microsoft Word and Excel. Also, it is no exaggeration to say that PDF builders such as LaTeX and Adobe Acrobat and presentation softwares such as Microsoft PowerPoint are indispensable for us, researchers. Our societies require capable human resources who can develop these softwares at the highest level possible, and various educational institutions such as uni-

[1]Corresponding Author: Shun HATTORI, School of Computer Science, Tokyo University of Technology, 1404–1 Katakura, Hachioji, Tokyo 192–0982, Japan; E-mail: hattori@cs.teu.ac.jp.

versities 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 Computer Science (CS) or Information Technology (IT) students/graduates. To conquer this problem, we launched a research project of Tangible Software Engineering Education [2], and have researched and developed various support systems in software engineering education to meet the needs of the times. In this paper, we propose a knowledge-dependent compiler with e-TA (electronic Teaching Agent) for not intermediate and advanced program developers but new learners of programming. And we show a prototype 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 and skills of users (learners).

A reason why not a few science and technology students (at least in Japan) hesitate or are weak in programming is certainly that more young Japanese are moving away from science and technology. But the main reason 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 and gcc are for not programming education but software development, and are for not beginners of programming but intermediate and advanced developers. Therefore, their error messages ask users for prior knowledge about programming and the programming language to some extent. However, because new learners of programming have not enough knowledge about programming itself as well as programming languages, they cannot easily be going on acquiring the knowledge without advice of others such as teachers and TAs (Teaching Assistants) just by using kindless error messages of conventional compilers. Static error messages of conventional compilers for program developers are dependent on only content of their source code. Such intelligent compilers to present easy-to-understand advice appropriately and on a timely basis (dynamically) depending on programming knowledge, skills and/or situation of learners who browse the compile errors for their source code, are not found as far as we know. The number of error messages for a source code is not rarely much more than the number of errors embedded in content of the source code, and several of them cannot give any help to even intermediate and advanced program developers. This is roughly the state of play in conventional compilers.

Our university, Tokyo University of Technology, offers Java programming introductory classes composed of classroom lectures and exercises to Computer Science (CS) freshmen. In every Java programming exercise, for some assignments by teachers, CS freshmen are instructed to edit their source code by such an editor as Emacs, to compile their source code by the most conventional Java compiler, javac [1], on the command line interface (CLI), and to modify and re-compile their source code if any error. By repeating such a process, they are expected to gradually acquire knowledge and skills of not only Java programming but also programming (software engineering) itself.

[2] http://www.teu.ac.jp/tangible/

There is no doubt that the unfriendliness and abstrusity of error messages by the most primary Java compiler, javac, cause many CS students who hesitate in programming (at least at our university). To lower this kind of barrier and decrease those students dropping out, one teacher and four TAs (Teaching Assistants) who are graduate students per classroom provide about 60 undergraduate students 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. But these human-powered supports have such limitations as follows:

- if not enough TAs for the number of learners in a classroom, the learners cannot be given advice appropriately and on a timely basis in case of bursts of their questions;
- if teachers give TAs not enough guidance about how to advise learners in advance, the learners cannot be given appropriate and uniform advice, because not only learners but also TAs (and teachers) have various level of knowledge and skills of (Java) programming, i.e., some TAs are enough high-level but others are not;
- some learners, even if stuck in compiler's error messages, cannot ask a teacher and/or TAs because of such a personality as shyness of strangers.

From the above-mentioned findings, we need a compiler oriented not software development by high-level users but software engineering education for low-level users, i.e., more intelligent compiler. In this paper, we propose a knowledge-dependent compiler with e-TA (electronic Teaching Agent) for software engineering education, and show a prototype of our education-oriented Java compiler, edu-javac, 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, as follows:

- using the other Java compiler as Eclipse Compiler for Java (ECJ) [2], GNU Compiler for Java (GCJ) [3], and Jikes [4].
- using such a GUI integrated development environment (IDE) as Eclipse and NetBeans [5] with keyword auto-complete (suggest) and high-level debug functions etc.;
- using such a programming language for new learners of programming as Nigari [6,7] and Kotodama [8,9];
- using such a programming (software engineering) environment for novices as PEN [10,11], Robotran [12], BlueJ [13], and Greenfoot [14].

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 command line interface (CLI), logically modify their source code with the help of compiler's error messages, and thus 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 re-compiling 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 which adopt javac as an internal Java compiler. Therefore, our proposed education-oriented Java compiler, edujavac, tries to generate advice for javac's error messages as appropriately and timely as possible.

1. Problems of Conventional Java Compilers and Programming Exercise Forms in Software Engineering Education

This section discusses problems of using the most conventional Java compiler, javac, in software engineering (programming) education and problems of programming exercise forms at our university, and extracts the requirements of our proposed education-oriented Java compiler, edujavac.

1.1. Problems of Java Compiler "javac"

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 most primary Java compiler, javac (in JDK1.6), offers the following 3 error messages for his source code with a typo "HJW.java (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
```

If such error messages by the most primary Java compiler, javac, were easy for low-level beginners as well as high-level developers to understand and modify them all alone, there would be no problem. But in fact, there are not a few CS students who get stuck in the error messages, i.e., they are not enough easy for at least new learners to understand and modify them all alone. In this case, the above-mentioned 3 error messages by the most 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 "HJW.java (model answer)" 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 confusing, because there is a typo in the 1st line of his source code "HJW.java (typo)" and the 2nd or 3rd error message points that there are some error in the 2nd or 4th line respectively. Offering such confusing 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 confusing error messages and to reduce its error messages to the number of errors embedded actually.

In this case of "HJW.java (typo)", a human teacher or TA would browse a model answer, a student's source code, and compiler's error messages, search for the error cause which is a typo of mistyping "`class`" to "`clas`" in the 1st line, and give "First, tackle the 1st error message." as the first advice to students. Subsequently, he would give "There is a typo anywhere.", "There is a typo in the 1st line.", "Replace `clas` with something in the 1st line.", and "Replace `clas` with `class` in the 1st line." as the last and most specific adivce to students.

Another typical error for the most introductory Java source code "HJW.java (model answer)" is forgetting to close a block by "`}`" as shown in "HJW.java (not-closed by `}`)". 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.

HJW.java (not-closed by })

```
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 (not-closed by `}`)".

javac HJW.java (not-closed by })

```
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 "First, indent properly." as more educational advice before giving "Insert `}` in the 5th line." as the last and most specific advice to students.

1.2. Problems of Programming Exercise Forms

Our university is entered by almost 600 Computer Science (CS) freshmen with very various levels of knowledge and skills of computer literacy and programming every year, and offers Java programming introductory classes composed of classroom lectures and exercises to the CS freshmen. While a teacher supports about 120 undergraduate students by using slides and/or textbooks per classroom in programming lectures, one teacher and four TAs (Teaching Assistants) who are graduate students support about 60 undergraduate students per classroom in programming exercises as shown in Figure 1. In every programming exercise, for some assignments given by a teacher, CS freshmen edit their source code by such an editor as Emacs, (re-)compile their source code by the most conventional

Java compiler, javac, on the command line interface (CLI), and finally have their source code checked by a TA with such a portable terminal as PSP or NDS. By repeating such a process, they are expected to gradually acquire knowledge and skills of not only Java programming but also programming (software engineering) itself. The current form of programming exercises has the following problems:

1. because the number of TAs are much less than the nmber of learners in a classroom (e.g., $4 + 1 \ll 60$), the learners cannot be always given advice appropriately and on a timely basis in case of bursts of their questions;
2. if teachers give TAs not enough guidance about how to advise learners in advance, the learners cannot be given appropriate and uniform advice, rather can be given wrong advice, because not only learners but also TAs (and teachers) have various level of knowledge and skills of (Java) programming, i.e., some TAs are enough high-level but others are not;
3. some learners, even if stuck in compiler's error messages, cannot ask a teacher and/or TAs because of such a personality as shyness of strangers and would leave them unsolved.

One approach to solve the current problems is to increase TAs as shown in Figure 2. It seems to be one ideal that learners can be always supported man-on-man by increasing TAs to the number of learners. Certainly, this approach would solve the 1st current problem. However, the other current problems cannot be solved. Rather, this approach would confound the 2nd current problem, i.e., becoming more difficult to give all TAs enough guidance about how to advise learners in advance, and cause the new problem, i.e., increasing personal costs.

Our proposed education-oriented Java compiler with knowledge-dependent e-TA (electronic Teaching Agent) can realize one ideal programming exercise form as shown in Figure 3 that learners can be always supported man-to-man by a computer-program e-TA instead of human TAs, i.e., without increasing TAs (personal costs), rather with eliminating some or all TAs. Our proposed education-

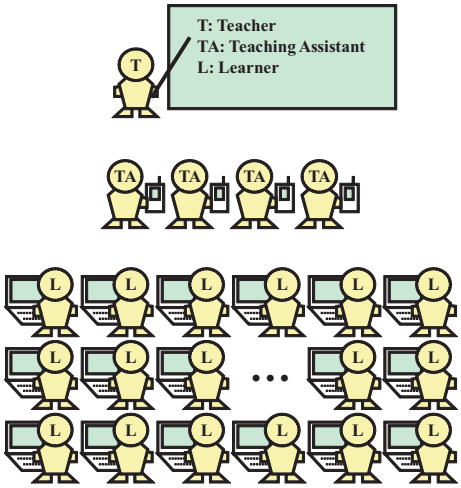

Figure 1. Exercise Form (Current).

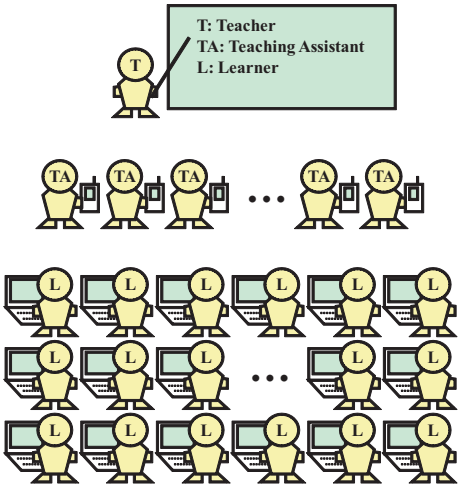

Figure 2. Exercise Form (Increase of TAs).

oriented Java compiler with e-TA requires such a new cost that each teacher specifies how to advise his students in a computer-understandable form in advance just as (rather instead of) giving TAs guidance about how to advise his students.

2. Design of Education-oriented Compiler

We now implement a prototype of education-oriented Java compiler, edujavac, in software engineering education for CS freshmen at our university. But this section describes a design of our proposed education-oriented compiler, in as general as possible regardless of our targeted programming language, Java, and various local requirements in the first year programming education at our university.

2.1. Objectives

Most of conventional compilers are basically designed for enough high-level experts of programming, and thus their error messages tend to be very difficult for learners of programming to understand (at least without training wheels). Unlike such conventional compilers, our proposed education-oriented compiler is designed for their neglected learners of programming, especially low-level new learners. It is probably fair to say that in the near future, anybody learns about programming languages to write a computer program for his wanted tasks on his own, like English as a language for world-wide communication. Therefore, we are in desperate need of a 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 (Teaching Assistant) but a computer e-TA (electronic Teaching Agent) of our proposed education-oriented

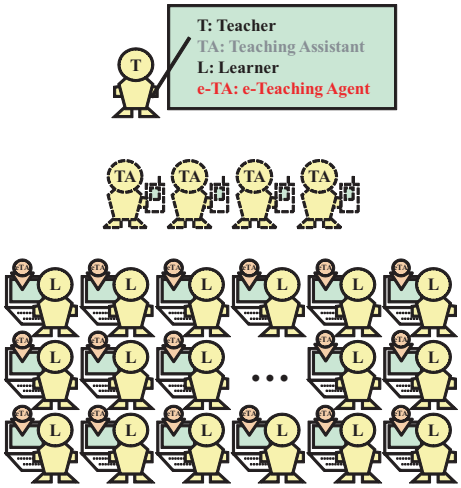

Figure 3. Programming Exercise Form (with e-TA).

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. The motivation of our proposed education-oriented compiler is not the optimal generation of an object code for a user-inputted source code. So, the conversion from a source code to an object code is entrusted to the existing compilers, and the e-TA of our proposed education-oriented compiler is responsible for the generation of easy-to-understand advice from compiler's error messages as appropriately and timely as possible. Note that the final goal of software engineering (programming) education is to allow learners to modify their source code on their own by using not our proposed e-TA's kind advice but the conventional compiler's unkind error messages. Therefore, excessive nurturing, e.g., always offering learners the most easy-to-understand (concrete and direct) advice like the answer of the optimal modification, is not good for the education (of course, may be good for the fastest development).

2.2. Requirements

The following requirements of (e-TA of) our proposed education-oriented compiler can be extracted by looking at the process that human teachers and TAs support their students in programming exercises.

- **Learner-dependency**:
  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) and skills of programming.
- **Teacher-dependency**:
  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. For example, a human teacher would like his TAs to advise his students a little more kindly if stuck in more than 15 minutes or if stuck in spite of much thinking and/or trying to modify on their own, and more kindly from 30 minutes to go in an exercise (total 90mins) to allow his students to work out their quick-fix source code. In addition, a human teacher would like his TAs to mask or supplement a student's unknown technical keywords in the compiler's error messages for his source code, what explanation and/or what textbooks and/or Web pages a human teacher would like his TAs to supplement each technical keyword with.
  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.

2.3. Basic Architecture

Figure 4 gives an overview of our 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 Teacher and Learner models.

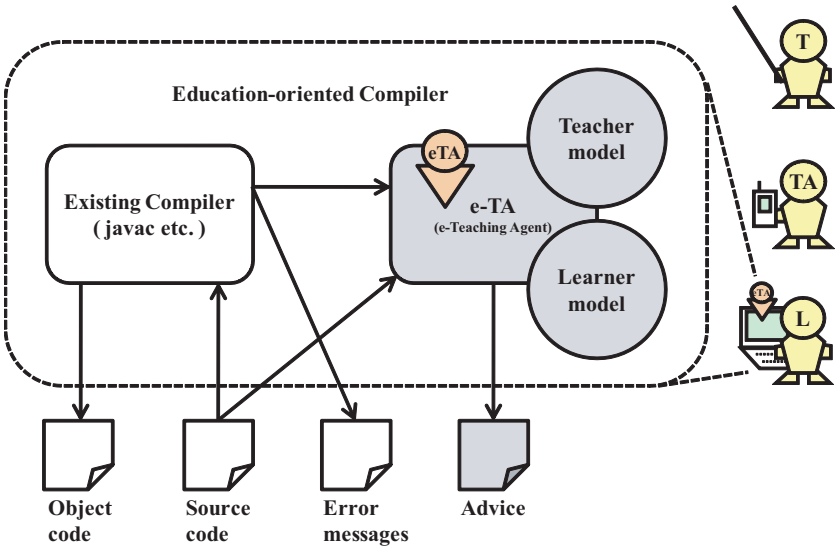

Figure 4. Overview of Education-oriented Compiler with e-TA.

3. Prototype of Knowledge-dependent Java Compiler with e-TA

As an implementation of our proposed education-oriented Java compiler, edu-javac, in software engineering (programming) education for CS freshmen at our university, this section shows a prototype of knowledge-dependent Java compiler with 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.

Each user (learner) has a knowledge directory, know-dir, on his local computer. Technical keywords with high frequency to some extent extracted from electronic documents in the user'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. In this prototype, only interval of updating content of e-TA's advice can be specified by each teacher.

The most primary Java compiler, javac, offers the quite same error messages indefinitely unless the source code inputted by the user is modified as below.

javac HJW.java (typo)

```
> javac 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

> javac 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
```

edujavac HJW.java (typo) [**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: 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
```

Our knowledge-dependent Java compiler 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) [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#Java

> 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#Java

> 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
```

edujavac HJW.java (typo) [**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 {
^
```

edujavac HJW.java (typo) [**supplement** except know-dir={"class", "interface"}]

```
> 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

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

> edujavac HJW.java
HJW.java:1: class, interface, or enum expected
clas HJW {
^
enum is a data type that ...
  TEXT: Objects First with Java, pp.233
   URL: http://en.wikipedia.org/wiki/Enumerated_type#Java

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

As above, our knowledge-dependent Java compiler with e-TA as an implementation of our proposed education-oriented Java compiler, edujavac, offers more easy-to-understand advice about the 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, as knowledge of his teacher.

As below, our knowledge-dependent Java compiler with e-TA offers the 1st advice about the source code with the lack of "}", HJW.java (not-closed by }), 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 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 (not-closed by }) [**supplement** except know-dir={"class"}]

```
> 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 used to format source 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 used to format source 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:4: insert } in the line no.5
```

Conclusions

In this paper, we have proposed a knowledge-dependent compiler with e-TA (electronic Teaching Agent) for software engineering education, and have shown a prototype 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.

In the near future, we try to evaluate our proposed education-oriented Java compiler, edujavac, by applying it to practical Java programming introductory classes for Computer Science (CS) freshmen at our university.

Acknowledgment

This work was supported in part by Tangible Software Engineering Education[3] (Project Leader: Taichi Nakamura, Tokyo University of Technology).

References

[1] Wikipedia - javac. http://en.wikipedia.org/wiki/Javac (2010).
[2] Eclipse. http://www.eclipse.org/ (2010).
[3] GCJ: The GNU Compiler for Java. http://gcc.gnu.org/java/ (2010).
[4] Jikes' Home. http://jikes.sourceforge.net/ (2010).
[5] Welcome to NetBeans. http://www.netbeans.org/ (2010).
[6] Nigari System. http://www2.eplang.jp/nigari/ (2010).
[7] S. Cho, M. Kai, A. Kawai, T. Hino, S. Maeshima, and K. Kakehi. 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).
[8] Kotodama. http://garuda.crew.sfc.keio.ac.jp/kotodamaCommunity/ (2010).
[9] A. Megumi, K. Okada, and H. 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).
[10] PEN (Programming Environment for Novices). http://grape.media.osaka-cu.ac.jp/PEN/ (2010).
[11] T. Nishida, A. Harada, R. Nakamura, Y. Miyamoto, and T. Matsuura. 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).
[12] R.M. Meyer and D.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).
[13] BlueJ - Teaching Java - Learning Java. http://www.bluej.org/ (2010).
[14] Greenfoot - The Java Object World. http://www.greenfoot.org/ (2010).

[3] http://www.teu.ac.jp/tangible/