JDK 1.5 Updates for
Introduction to Java Programming with SUN ONE Studio 4

NOTE: SUN ONE Studio is almost identical with NetBeans. NetBeans is open source and can be downloaded from www.netbeans.org. I recommend you use NetBeans to replace SUN ONE Studio.

1 Introduction

There are already more features in Java than an introductory course can cover. This update is not aimed to cover all the new features in JDK 1.5. Nevertheless, some simple features in JDK 1.5 can be incorporated to simplify programming. This update presents the following JDK 1.5 features in modules that can be inserted into the current text:

- To format output using printf.
- To use the Scanner class to simplify console input.
- To simplify programming using enhanced for loops.
- To use the Scanner class to scan tokens using words as delimiters.
- To simplify programming using JDK 1.5 automatic conversion between primitive types and wrapper class types.
- To simplify programming using JDK 1.5 generic types.

2 Compiling JDK 1.5 Code in NetBeans

To compile JDK 1.5 code, you need NetBeans 3.6 or higher. You can download NetBeans from www.netbeans.org.

The default compiler in NetBeans is JDK 1.4. To use a JDK 1.5 compiler, choose Tools, Options to open the Options dialog box as shown in Figure 1. Choose Building, Compiler Types, and External Compilation under the Options and click the eclipses in the value field for External Compiler to display the External Compilation dialog box, as shown in Figure 2. Add -source 1.5 in the Arguments just before -classpath exactly as shown in Figure 2.

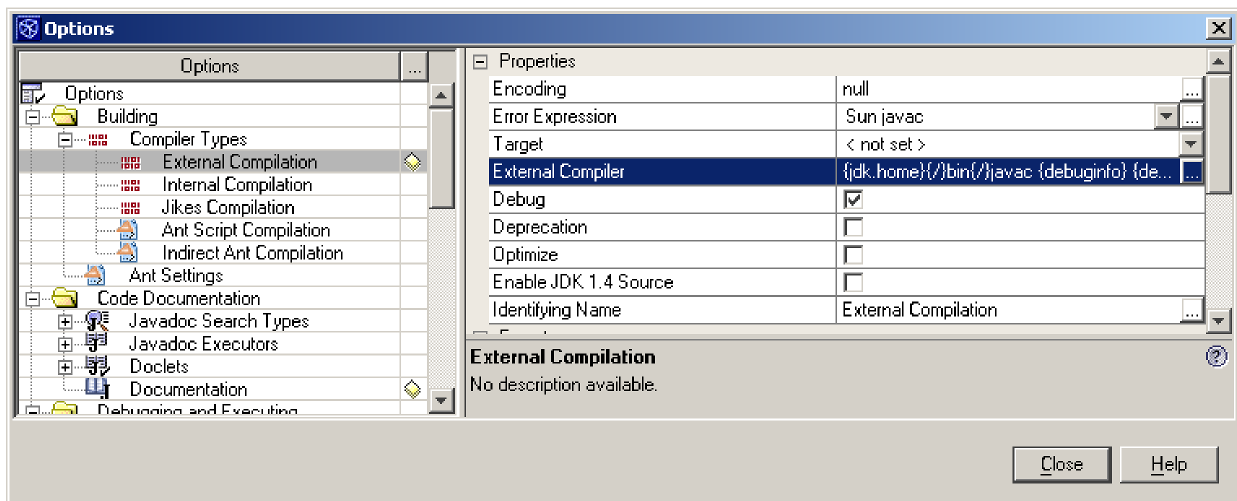


Figure 1

The Options dialog enables you to configure NetBeans.

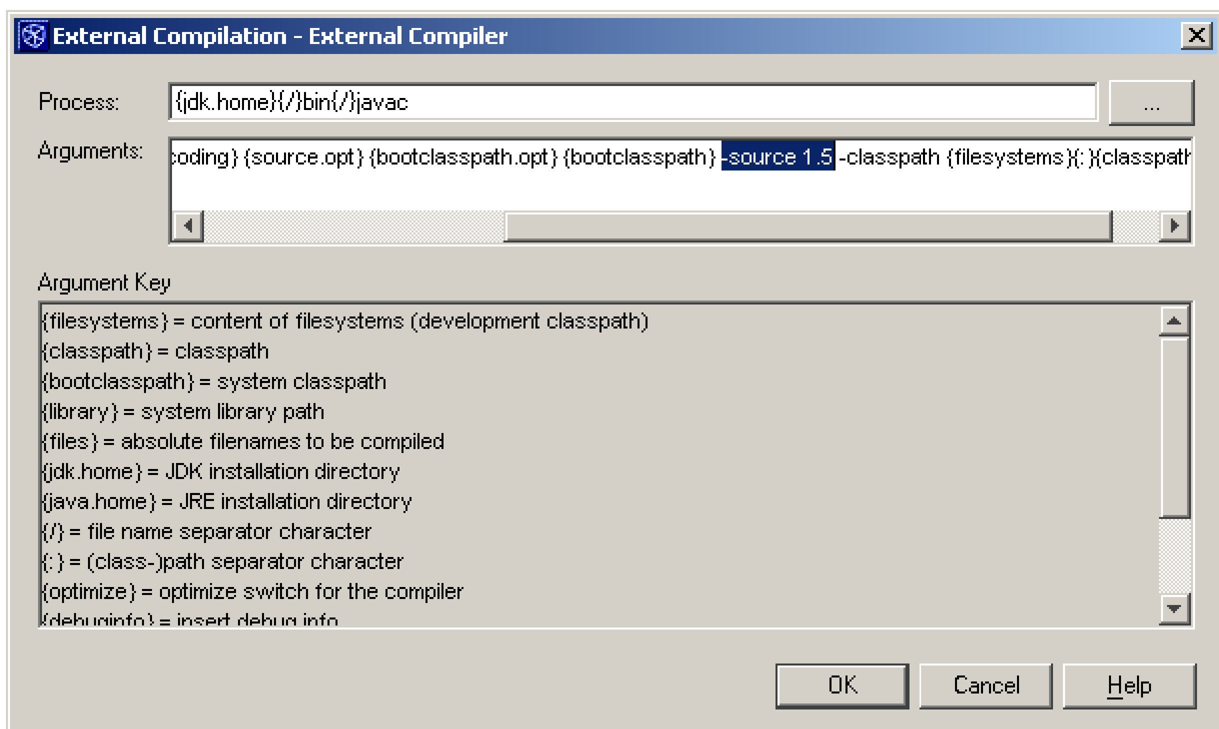


Figure 2

Adding -source in the compiler arguments enables you to compile JDK 1.5 code.

NOTE: By default, the **javac** compiler is for JDK 1.4. To compile JDK 1.5 code (e.g., Test.java) from the command line, use the following command:

```
javac -source 1.5 Test.java
```

3 Formatting Output

> NOTE: You can insert this section before Section 2.14, "Case Studies."

You already know how to display console output using the print or println methods. JDK 1.5 introduced a new printf method that enables you to format output. The syntax to invoke this method is

```
System.out.printf(format, items)
```

Where format is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign. Table 1 lists some frequently-used specifiers:

Table 1

Frequently-used specifiers

Specifier	Output	Example
%b	a boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s	a string	"Java is cool"

Here is an example:

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);

display          count is 5 and amount is 45.560000
```

items

Items must match the specifiers in order, in number, and in exact type. For example, the specifier for count is %d, for (1 < 0) is %b, and for amount is %f. By default, a floating-point value is displayed with six digits after the decimal point. You can specify the width and precision in a specifier, as shown in the examples in Table 2.

Table 2

Examples on Specifying Width and precision

Example	Output
%5c	Output the character and add four spaces before the character item.
%6b	Output the boolean value and add one space before the false value and two spaces before the true value.
%5d	Output the integer item with width at least 5. If the number of digits in the item is < 5, add spaces before the number.
%10.2d	Output the floating-point item with width at least 10 including a decimal point and two digits after the point. So there are 7 digits allocated before the decimal point. If the number of digits before the decimal in the item is < 7, add spaces before the number.

%10.2e	Output the floating-point item with width at least 10 including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than 10, add spaces before the number.
%12s	Output the string with width at least 12. If the string item has less than 12 characters, add spaces before the string.

CAUTION: The items must match the specifiers in exact type. The item for the specifier %f or %e must be a floating-point type value such as 40.0, not 40. So, an int variable cannot match %f or %e.
TIP: The % sign denotes a specifier. To output a literal % in the format string, use %%.

4 Simplifying Console Input Using Scanner

NOTE: You can insert this section before Section 2.14, "Case Studies."

The text uses JOptionPane to obtain input from dialog boxes. Alternatively, you can use the new JDK 1.5 Scanner class to obtain input from the console.

Java uses System.out to refer to standard output device and System.in to standard input device. By default the output device is the console and the input device is the keyboard. To perform console output, you simply use the print or println method to display a primitive value or a string to the console. Keyboard input is not directly supported in Java, but you can use the Scanner class to create an object to read input from System.in as follows:

```
Scanner scanner = new Scanner(System.in);
```

Now you can use next(), nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or nextBoolean() to obtain to a string, byte, short, int, long, float, double, or boolean value. For example, the following statements prompt the user to enter a double value from the console.

```
System.out.print("Enter a double value: ");
Scanner scanner = new Scanner(System.in);
double d = scanner.nextDouble();
```

The following is a complete example that reads various types of data from the console using the Scanner class. A sample run of this program is shown in Figure 1.

```
import java.util.Scanner; // Scanner is in java.util

public class TestScanner {
  public static void main(String args[]) {
    // Create a Scanner
    Scanner scanner = new Scanner(System.in);

    // Prompt the user to enter an integer
    System.out.print("Enter an integer: ");
    int intValue = scanner.nextInt();
    System.out.println("You entered the integer " + intValue);

    // Prompt the user to enter a double value
    System.out.print("Enter a double value: ");
```

```
        double doubleValue = scanner.nextDouble();
        System.out.println("You entered the double value "
          + doubleValue);

        // Prompt the user to enter a string
        System.out.print("Enter a string without space: ");
        String string = scanner.next();
        System.out.println("You entered the string " + string);

        // Prompt the user to enter a boolean
        System.out.print("Enter a boolean: ");
        boolean booleanValue = scanner.nextBoolean();
        System.out.println("You entered the boolean " + booleanValue);
    }
}
```

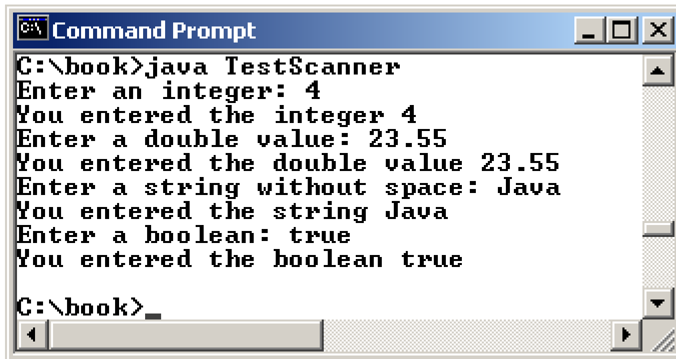


Figure 1

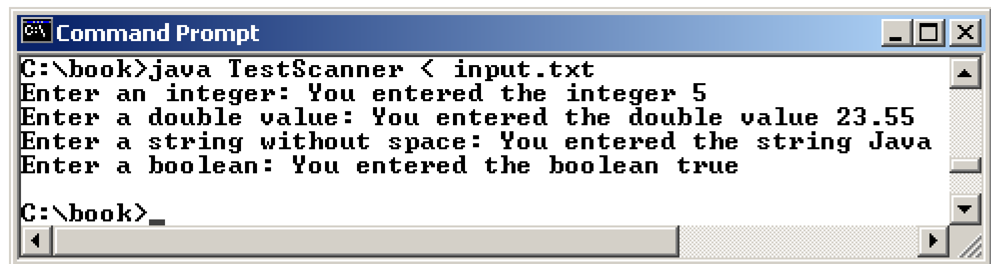
You can enter input from a command window.

TIP

One benefit of using the console input is that you can store the input values in a text file and pass the file from the command line using the following command:

```
java TestScanner < input.txt
```

where input.txt is a text file that contains the data, as shown in (A) in Figure 2. The output of java TestScanner < input.txt is shown in (B) in Figure 2.



(A) (B)

Figure 2

(A) You can create a text file using NotePad. (B) The data in the text file is passed to the program.

You can also save the output into a file using the following command:

```
java TestScanner < input.txt > out.txt
```

5 Enhanced for Loop

NOTE: You can insert this section before Section 5. 4, "Passing Arrays to Methods."

JDK 1.5 introduced a new for loop that enables you to traverse the complete array sequentially without using an index variable. For example, the following code displays all elements in the array myList:

```
for (double value: myList)
  System.out.println(value);
```

In general, the syntax is

```
for (elementType value: arrayRefVar) {
  // Process the value
}
```

You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

6 Extracting Tokens Using Scanner

NOTE: You can insert this section before Section 7.6, "Command-Line Arguments."

The delimiters are single characters in StringTokenizer. You can use the new JDK 1.5 java.util.Scanner class to specify a word as a delimiter. Here is an example that uses the word *Java* as a delimiter to scan tokens in a string:

```
String s = "Welcome to Java! Java is fun! Java is cool!";
Scanner scanner = new Scanner(s);
scanner.useDelimiter("Java");

while (scanner.hasNext())
  System.out.println(scanner.next());
```

Line 2 creates an instance of Scanner using the static create(String) method. Line 3 sets "Java" as a delimiter. Line 5, hasNext() returns true if there are still more tokens left. Line 6, the next() method returns a token as a string. So, the output from this code is

```
Welcome to
!
 is fun!
 is cool!
```

If a token is a primitive data type value, you can use the methods nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or nextBoolean() to obtain it. For example, the following code adds all numbers in the string. Note that the delimiter is space by default.

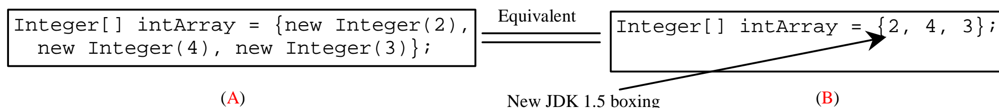
```
String s = "1 2 3 4";
Scanner scanner = new Scanner(s);

int sum = 0;
while (scanner.hasNext())
  sum += scanner.nextInt();

System.out.println("Sum is " + sum);
```

NOTE: StringTokenizer can specify several single characters as delimiters. Scanner can use a single character or a word as the delimiter. So, if you need to scan a string with multiple single characters as delimiters, use StringTokenizer. If you need to use a word as the delimiter, use Scanner.

7 Automatic Conversion Between Primitive Types and Wrapper Class Types

NOTE: You can insert this section before Section 9.7, "Case Studies."

JDK 1.5 allows primitive type and wrapper classes to be converted automatically. For example, the following statement in (A) can be simplified as in (B):



Assigning a primitive value to a wrapper object is called *boxing*. Assigning a wrapper object to a primitive type value is called *unboxing*, as shown in the following example:

```
Integer[] intArray = {1, 2, 3};
System.out.println(intArray[0] + intArray[1] + intArray[2]);
```

intArray[0], intArray[1], and intArray[2] are automatically converted int values and these values are added together.

8 Simplifying Traversing Arrays Using Enhanced for Loops

NOTE: You can insert this section in Section 9.7, "Case Studies."

You can simplify the code in Lines 19-25 in Example 19.1 using a JDK 1.5 enhanced for loop without using an iterator, as follows:

```
for (Object element: set)
  System.out.print(element.toString() + " ");
```

9 Using Generic Types

NOTE: You can insert this section after Section 19.8, "The Vector and Stack Classes."

You can add any object into a collection (i.e., set, list, vector, or stack). Sometimes, you wish only one type of objects to be in a collection. The new JDK 1.5 generic types provide a mechanism to support type check at compile time. For example, the following statement creates a set for strings:

```
HashSet<String> set = new HashSet<String>();
```

You can now add only strings into the set. For example,

```
set.add("Red");
```

If you attempt to add a non-string, a compile time error would occur. For example, the following statement is now illegal, because set can contain strings only.

```
set.add(new Integer(1));
```

To retrieve a value from a collection with a specified element type, no casting is needed because the compiler already knows the element type. For example, the following statements create a list that contains double values only, add elements to the list, and retrieve elements from the list.

```
ArrayList<Double> list = new ArrayList<Double>();
list.add(5.5); // 5.5 is automatically converted to new Double(5.5)
list.add(3.0); // 3.0 is automatically converted to new Double(3.0)
Double doubleObject = list.get(0); // No casting is needed
double d = list.get(1); // Automatically converted to double
```

In Lines 2 and 3, 5.5 and 3.0 are automatically converted into Double objects and added to list. Automatic conversion is a new feature in JDK 1.5, which was introduced in Section 9.6. In Line 4, the first element in list is assigned to a Double variable. No casting is necessary since list is declared for Double objects. In Line 5, the second element in list is assigned to a double variable. The object in list.get(1) is automatically converted into a primitive type value.

NOTE

All the collection classes support generic types. So, you can use the <> notation to specify a

particular type for the elements in a collection. To enable a class to support generic types, a class has to be declared using a special syntax. Supplement P, "Creating Generic Types," discusses how to create generic types.