Think Java

CHAPTER 3: INPUT AND OUTPUT

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- The programs we've looked at so far simply display messages, which doesn't really involve that much computation.
- This chapter will show you how to read input from the keyboard, use that input to calculate a result, and then format that result for output.

Topic

- Class as the basic program unit.
- Elements and the Structure of a Program
- I/O
- Basic Arithmetic Programming

LECTURE 1

The System class

Demo: Simple Circle Class

CIRCLE.JAVA

The System class

- **System** is a class that provides methods related to the "system" or environment where programs run. It also provides **System.out**, which is a special value that has additional methods (like println) for displaying output.
- In fact, we can use System.out.println to display the value of **System.out**:

```
System.out.println(System.out);
```

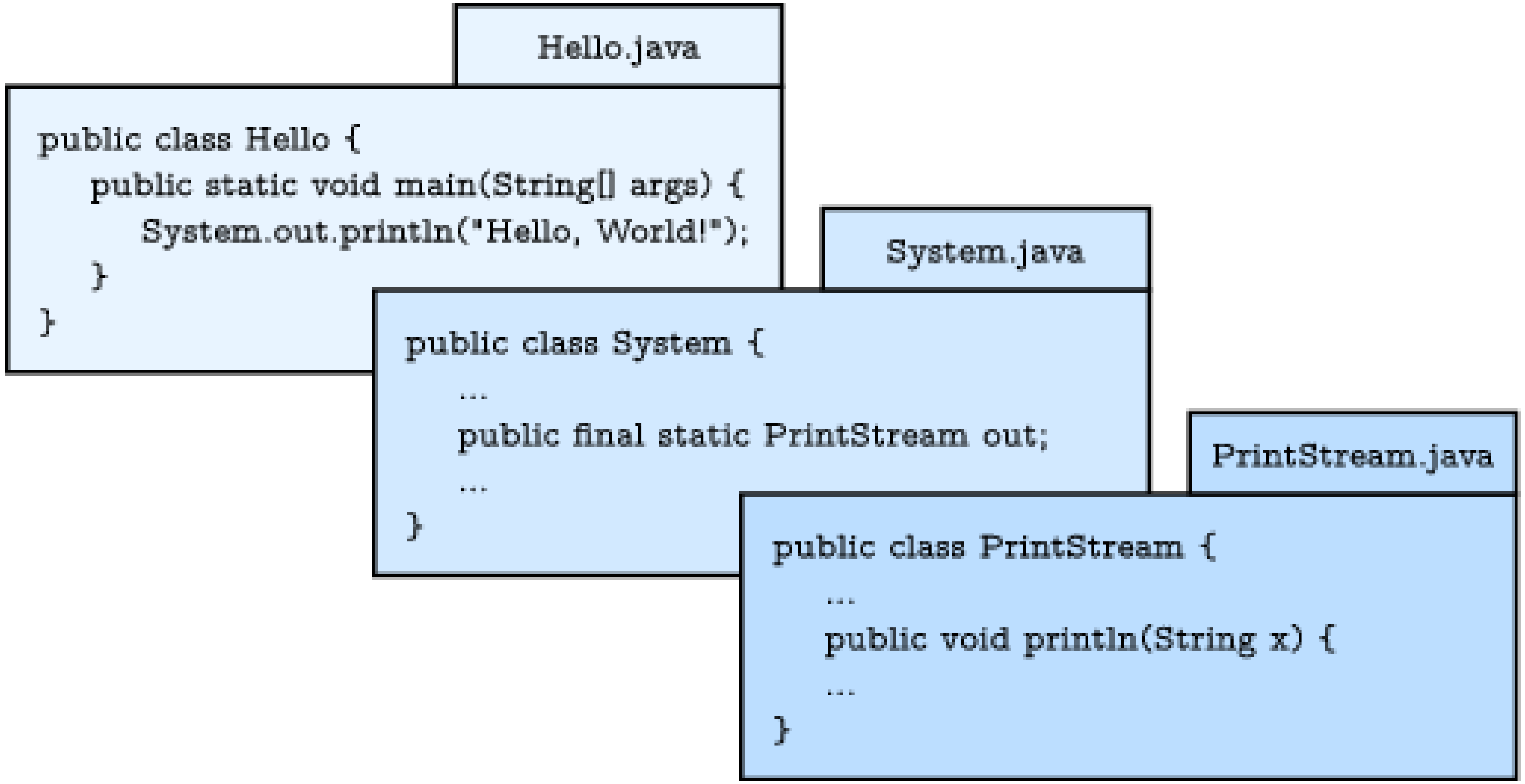
- The result is:

```
java.io.PrintStream@685d72cd
```

- This output indicates that **System.out** is a **PrintStream**, which is defined in a package called java.io. A package is a collection of related classes; java.io contains classes for "I/O" which stands for input and output.

System Class

- The numbers and letters after the @ sign are the **address** of **System.out**, represented as a hexadecimal (base 16) number. The address of a value is its location in the computer's memory, which might be different on different computers. In this example the address is **685d72cd**, but if you run the same code you will likely get something else.
- As shown in Figure 3.1, System is defined in a file called **System.java**, and **PrintStream** is defined in **PrintStream.java**. These files are part of the Java **library**, which is an extensive collection of classes that you can use in your programs. The source code for these classes is usually included with the compiler (see Section 10.6).



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

```
public class System {
    ...
    public final static PrintStream out;
    ...
}
```

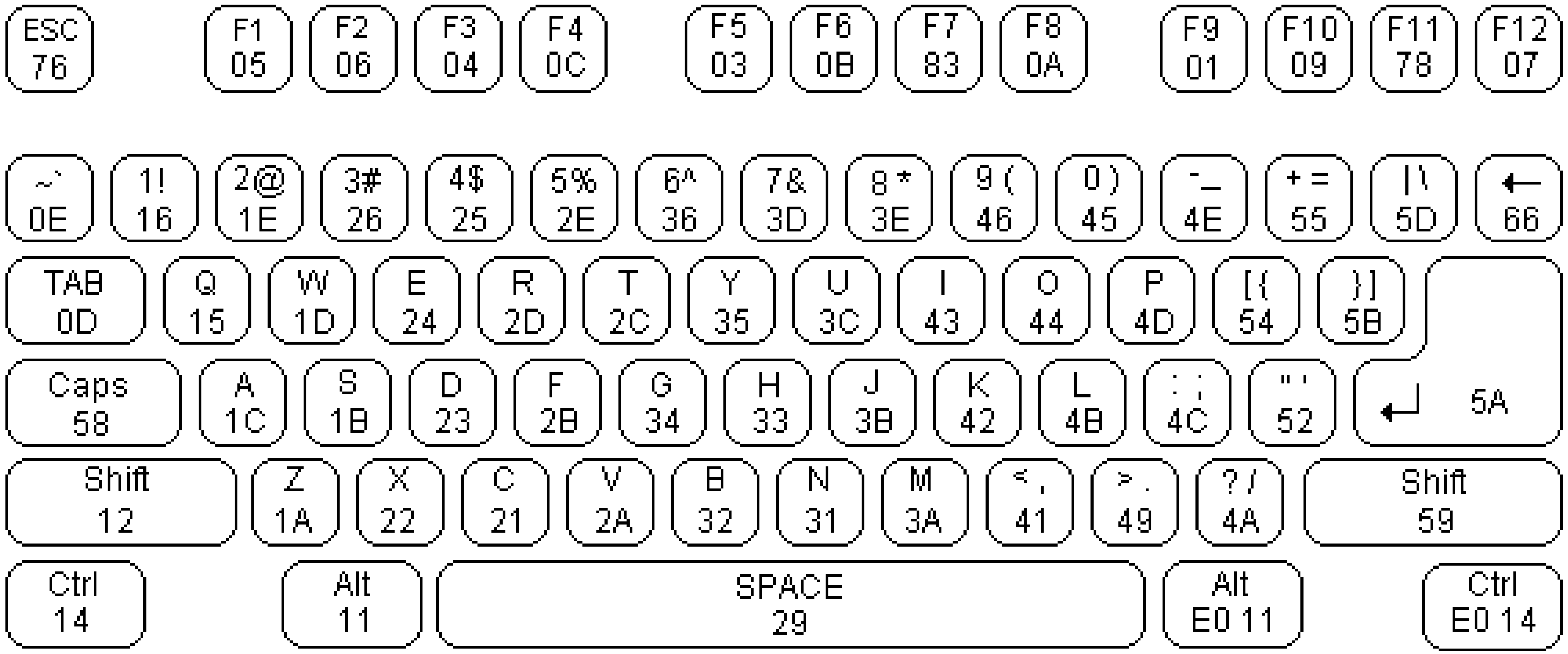
```
public class PrintStream {
    ...
    public void println(String x) {
    ...
}
```

Figure 3.1: System.out.println refers to the out variable of the System class, which is a PrintStream that provides a method called println.

LECTURE 2

The Scanner class

Keyboard Scan Code



System.in

- The System class also provides the special value **System.in**, which is an **InputStream** that has methods for reading input from the keyboard. These methods are not convenient to use, but fortunately Java provides other classes that make it easy to handle common input tasks.

- For example, **Scanner** is a class that provides methods for inputting words, numbers, and other data. **Scanner** is provided by **java.util**, which is a package that contains various “utility classes”. Before you can use **Scanner**, you have to import it like this:

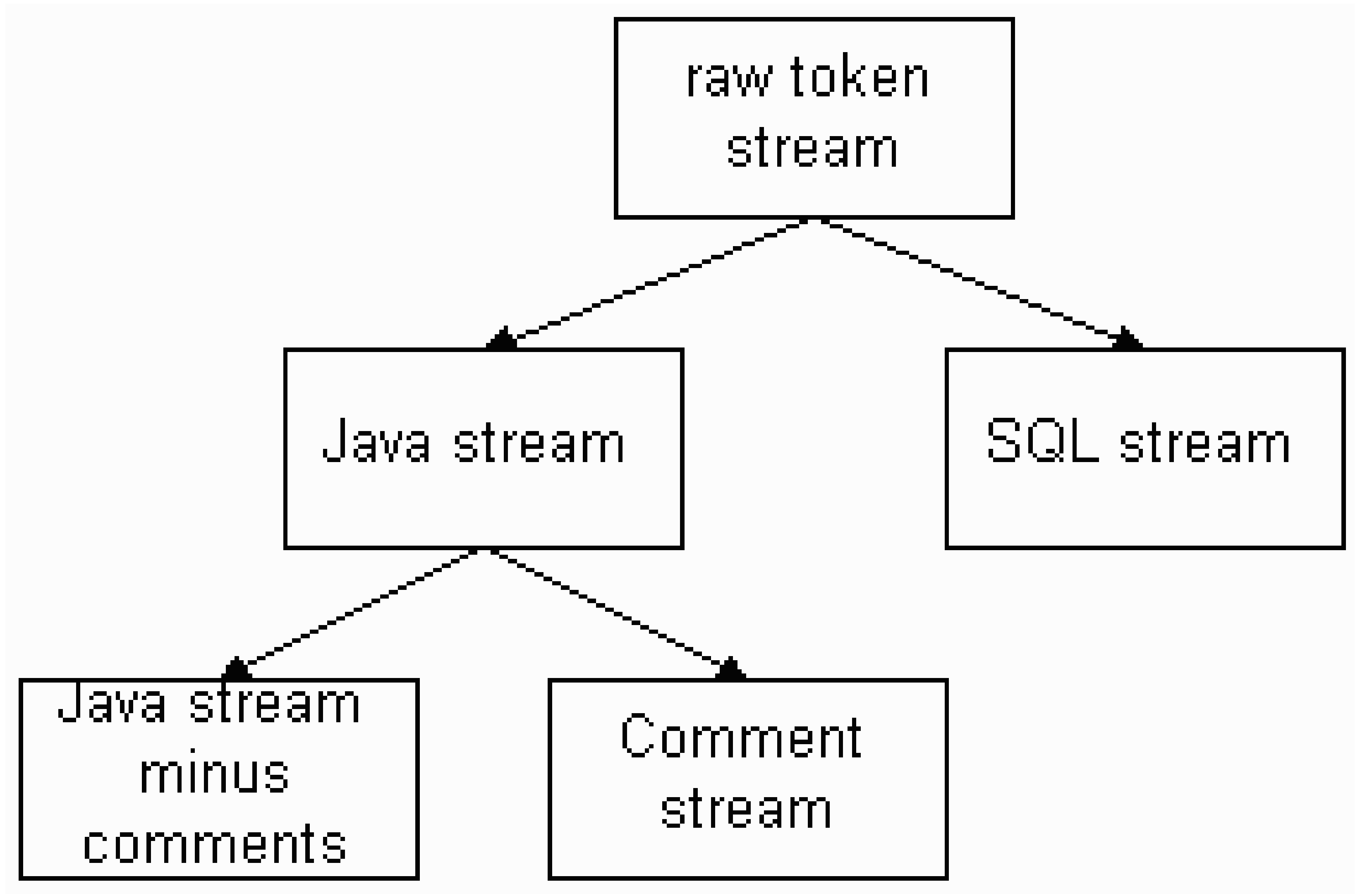
```
  import java.util.Scanner;
```

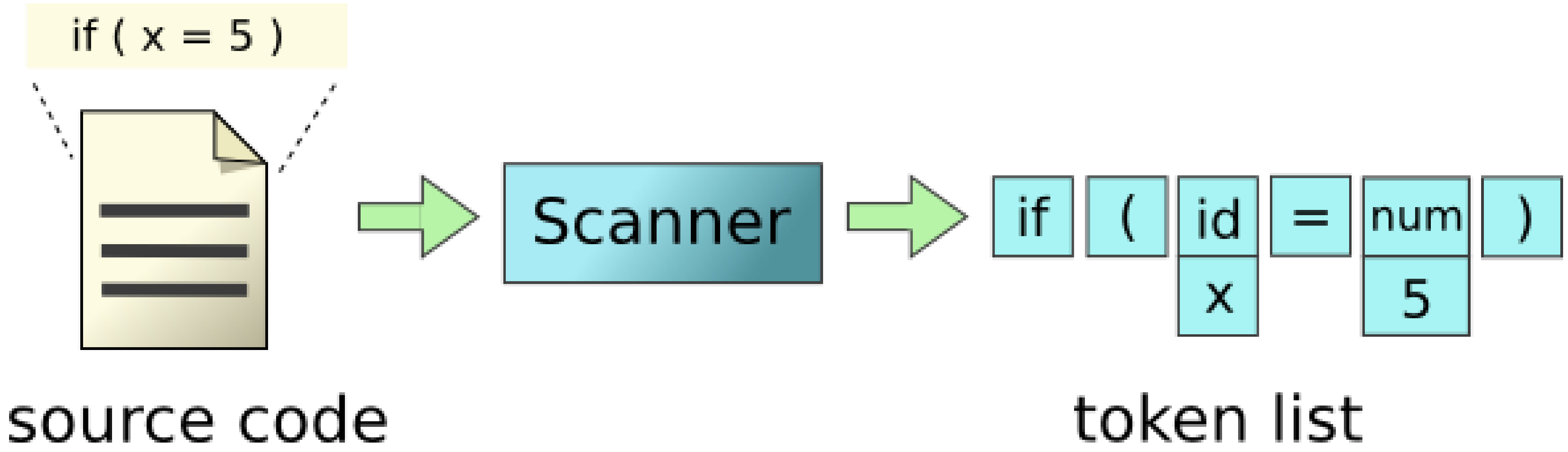
`import` Statement

This **import** statement tells the compiler that when you refer to Scanner, you mean the one defined in **java.util**. Using an import statement is necessary because there might be another class named Scanner in another package. Import statements can’t be inside a class definition. By convention, they are usually at the beginning of the file.

Next you have to initialize the **Scanner**. This line declares a Scanner variable named in and creates a Scanner that reads input from System.in:

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


raw token stream
Java stream
SQL stream
Java stream minus comments
Comment stream


if (x = 5)
source code
Scanner
if
(
id
x
=
num
5
)
token list

In-Class Demo Program

- System.in
- System.out

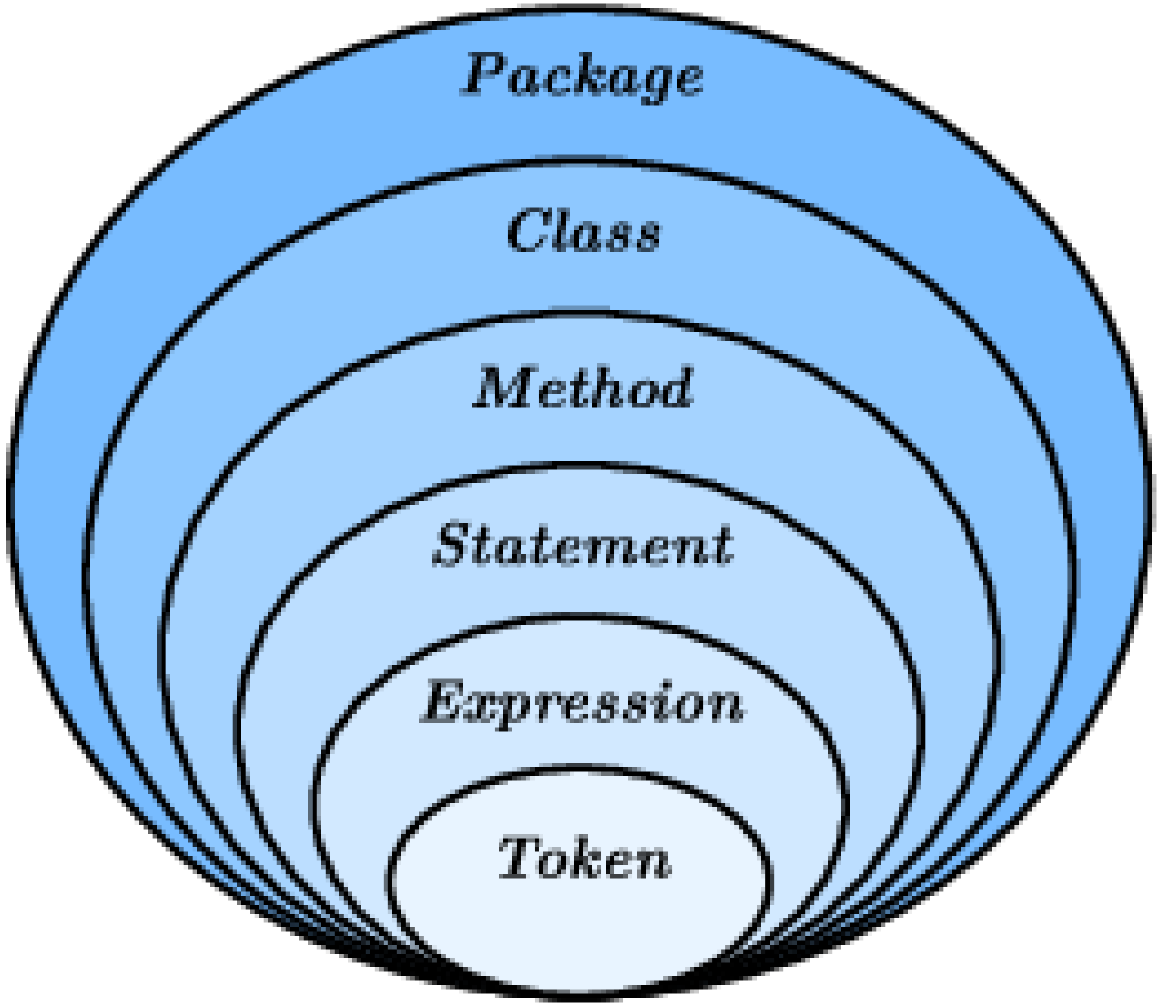
ECHO.JAVA

LECTURE 3

Language elements

Language Elements

- Java applications are typically organized into packages (like **java.io** and **java.util**) that include multiple classes (like **PrintStream** and **Scanner**). Each class defines its own methods (like **println** and **nextLine**), and each method is a sequence of statements.
- Each statement performs one or more computations, depending on how many expressions it has, and each expression represents a single value to compute. For example, the assignment statement hours = minutes / 60.0; contains a single expression: minutes / 60.0.



java.util

Scanner

nextInt

hour = in.nextInt();

hour * 60

hour

Tokens

- **Tokens** are the most basic elements of a program, including numbers, variable names, operators, keywords, parentheses, braces, and semicolons. In the previous example, the tokens are hours, =, minutes, /, 60.0, and ; (spaces are ignored by the compiler).
- Knowing this terminology is helpful, because error messages often say things like “**not a statement**” or “**illegal start of expression**” or “**unexpected token**”. Comparing Java to English, statements are complete sentences, expressions are phrases, and tokens are individual words and punctuation marks.

Tokens

- Note there is a big difference between the Java **language**, which defines the elements in Figure 3.2, and the Java **library**, which provides the built-in classes that you can import. For example, the keywords public and class are part of the Java language, but the names PrintStream and Scanner are not.

LECTURE 4

Literals and constants

Data Input

- We can write a program to help. We'll use a Scanner to input a measurement in inches, convert to centimeters, and then display the results. The following lines declare the variables and create the Scanner:

```
int inch;
double cm;
Scanner in = new Scanner(System.in);
```

- The next step is to prompt the user for the input. We'll use print instead of println so they can enter the input on the same line as the prompt. And we'll use the Scanner method nextInt, which reads input from the keyboard and converts it to an integer:

```
System.out.print("How many inches? ");
inch = in.nextInt();
```

Data Output

- Next we multiply the number of inches by 2.54, since that’s how many centimeters there are per inch, and display the results:

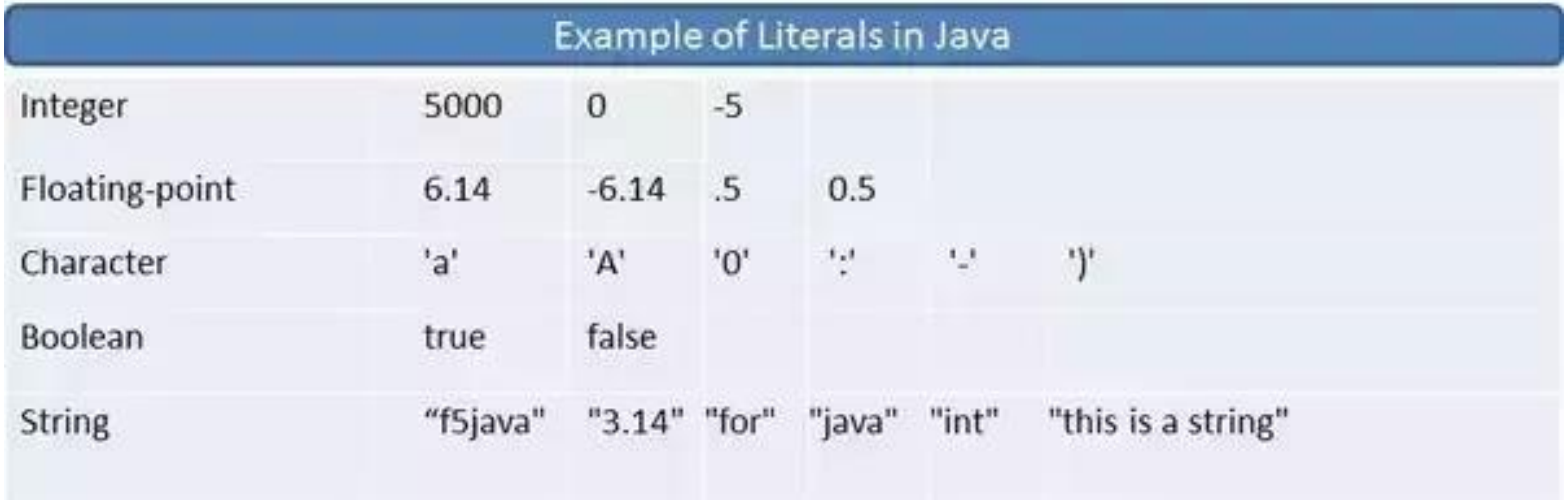
```
cm = inch * 2.54;
System.out.print(inch + " in = ");
System.out.println(cm + " cm");
```

- This code works correctly, but it has a minor problem. If another programmer reads this code, they might wonder where 2.54 comes from. For the benefit of others (and yourself in the future), it would be better to assign this value to a variable with a meaningful name.

Data Literals

- A value that appears in a program, like the number 2.54, is called a literal. In general, there's nothing wrong with literals. But when numbers like 2.54 appear in an expression with no explanation, they make the code hard to read. And if the same value appears many times and could change in the future, it makes the code hard to maintain.
- Values like 2.54 are sometimes called magic numbers (with the implication that being "magic" is not a good thing). A good practice is to assign magic numbers to variables with meaningful names, like this:

```
double cmPerInch = 2.54;
cm = inch * cmPerInch;
```



Example of Literals in Java

Integer	5000	0	-5			
Floating-point	6.14	-6.14	.5	0.5		
Character	'a'	'A'	'0'	':'	'-'	')'
Boolean	true	false				
String	“f5java"	"3.14"	"for"	"java"	"int"	"this is a string"

Constant

- This version is easier to read and less error-prone, but it still has a problem. Variables can vary (hence the term), but the number of centimeters in an inch does not. Once we assign a value to cmPerInch, it should never change. Java provides the keyword final, a language feature that enforces this rule.

```
    final double CM_PER_INCH = 2.54;
```

- Declaring that a variable is final means that it cannot be reassigned once it has been initialized. If you try, the compiler gives an error. Variables declared as final are called constants. By convention, names for constants are all uppercase, with the underscore character (_) between words.

LECTURE 5

Formatting output

Output of double data type

- When you output a double using print or println, it displays up to 16 decimal places:

```
System.out.print(4.0 / 3.0);
```

- The result is:

```
1.3333333333333333
```

- That might be more than you want. System.out provides another method, called printf, that gives you more control of the format. The “f” in printf stands for “formatted”. Here’s an example:

```
System.out.printf("Four thirds = %.3f", 4.0 / 3.0);
```

Formatted Output

- The first value in the parentheses is a format string that specifies how the output should be displayed. This format string contains ordinary text followed by a format specifier, which is a special sequence that starts with a percent sign. The format specifier \%.3f indicates that the following value should be displayed as floating-point, rounded to three decimal places. The result is:
- Four thirds = 1.333
- The format string can contain any number of format specifiers; here's an example with two of them:

```
int inch = 100;
double cm = inch * CM_PER_INCH;
System.out.printf("%d in = %f cm\n", inch, cm);
```

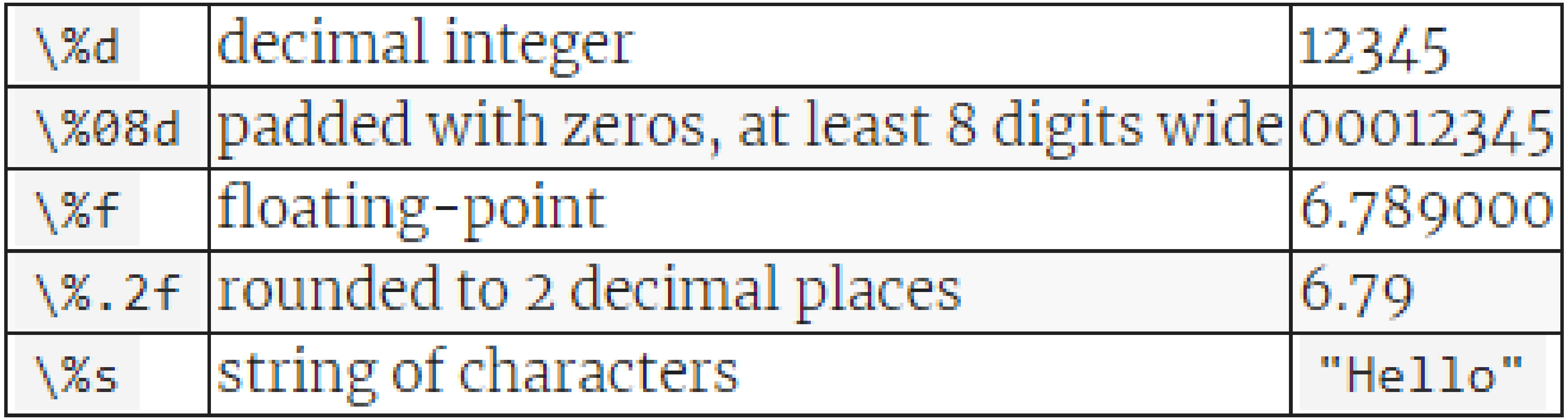
Formatted Output

- The result is:

```
100 in = 254.000000 cm
```

- Like print, printf does not append a newline. So format strings often end with a newline character.
- The format specifier \%d displays integer values (“d” stands for “decimal”, not double). The values are matched up with the format specifiers in order, so inch is displayed using \%d, and cm is displayed using \%f.
- Learning about format strings is like learning a sub-language within Java. There are many options, and the details can be overwhelming. Table 3.1 lists a few common uses, to give you an idea of how things work.

Format String



\%d	decimal integer	12345
\%08d	padded with zeros, at least 8 digits wide	00012345
\%f	floating-point	6.789000
\%.2f	rounded to 2 decimal places	6.79
\%s	string of characters	"Hello"

Formatted Output

- For more details, refer to the documentation of java.util.Formatter. The easiest way to find documentation for Java classes is to do a web search for “Java” and the name of the class.
- In contrast to print and println, printf uses commas to separate each value. The following line accidentally concatenates inch and cm to the format string, rather than substitute them:

```
System.out.printf("%d in = %f cm\n" + inch + cm);  // error
```

Formatted Output

- Unfortunately the compiler won't catch this kind of bug, because it's within a legal Java statement. However when you run the program, it will display:
- Exception in thread "main" java.util.MissingFormatArgumentException:
- Format specifier '%d'

```
	at java.util.Formatter.format(Formatter.java:2519)
	at java.io.PrintStream.format(PrintStream.java:970)
	at java.io.PrintStream.printf(PrintStream.java:871)
	at Example.main(Example.java:10)
```

Formatted Output

- Error messages may seem cryptic, but it’s important to read them carefully. Starting from the top, this one says “missing format argument” and “format specifier %d”. In order words, it doesn’t know what value to substitute for the %d. The bottom of the error indicates where to look: Example.java line 10.
- It might be difficult to see what’s wrong, given that inch and cm are at the end of the printf statement. But if inch is 1 and cm is 2.54, the actual format string would be "\%d in = \%f cm\\n12.54"

LECTURE 6

Type cast operators

Type Casting

- Now suppose we have a measurement in centimeters, and we want to round it off to the nearest inch. It is tempting to write:

```
  inch = cm / CM_PER_INCH;   // syntax error
```

- But the result is an error – you get something like, “incompatible types: possible lossy conversion from double to int.” The problem is that the value on the right is floating-point, and the variable on the left is an integer.
- Java converts an int to a double automatically, since no information is lost in the process. On the other hand, going from double to int would lose the decimal places. Java doesn’t perform this operation automatically in order to ensure that you are aware of the loss of the fractional part of the number.

Explicit Casting

- The simplest way to convert a floating-point value to an integer is to use a type cast, so called because it molds or “casts” a value from one type to another. The syntax for type casting is to put the name of the type in parentheses and use it as an operator.

```
double pi = 3.14159;
int x = (int) pi;
```

- The (int) operator has the effect of converting what follows into an integer. In this example, x gets the value 3. Like integer division, casting to an integer always rounds toward zero, even if the fraction part is 0.999999 (or -0.999999). In other words, it simply throws away the fractional part.

Explicit Casting

- In order to use a cast operator, the types must be compatible. For example, you can’t cast a String to an int because a string is not a number.

```
String str = "3";
int x = (int) str;  // error: incompatible types
```

- Type casting takes precedence over arithmetic operations. In the following example, the value of pi gets converted to an integer before the multiplication.

```
double pi = 3.14159;
double x = (int) pi*20.0;//result is 60.0, not 62.0
```

Casting

- Keeping that in mind, here's how we can convert centimeters to inches:

```
inch = (int) (cm / CM_PER_INCH);
System.out.printf("%f cm = %d in\n", cent, inch);
```

- The parentheses after the cast operator require the division to happen before the type cast. And the result is rounded toward zero. We will see in the next chapter how to round floating-point numbers to the closest integer

LECTURE 7

Remainder operator

Integer Division and Modulus

- Let’s take the example one step further: suppose you have a measurement in inches and you want to convert to feet and inches. The goal is divide by 12 (the number of inches in a foot) and keep the remainder.
- We have already seen the **division** operation (/), which computes the quotient of two numbers. If the numbers are integers, it performs integer division. Java also provides the modulo operation (\%), which divides two numbers and computes the remainder.

Integer Division and Modulus

- Using division and modulo, we can convert to feet and inches like this:

```
feet = 76 / 12;       // quotient
inches = 76 % 12;     // remainder
```

- The first line yields 6. The second line, which is pronounced "76 mod 12", yields 4. So 76 inches is 6 feet, 4 inches.

Integer Division and Modulus

- Many people (and textbooks) incorrectly refer to \% as the "modulus operator". In mathematics, however, modulus is the number you're dividing by. In the previous example, the modulus is 12. The Java language specification refers to \% as the "remainder operator".
- The remainder operator looks like a percent sign, but you might find it helpful to think of it as a division sign (÷) rotated to the left.

Integer Division and Modulus

- Modular arithmetic turns out to be surprisingly useful. For example, you can check whether one number is divisible by another: if x \% y is zero, then x is divisible by y. You can use remainder to “extract” digits from a number: x \% 10 yields the rightmost digit of x, and x \% 100 yields the last two digits.
- And many encryption algorithms use the remainder operator extensively.

LECTURE 8

Putting it all together

Features in this Sample Program

(1) import Java library classes,

(2) create a Scanner,

(3) get input from the keyboard,

(4) format output with printf, and

(5) divide and mod integers.

In-Class Demo Program

- import Java library classes,
- create a Scanner,
- get input from the keyboard,
- format output with printf, and
- divide and mod integers.

COVERT.JAVA

Metric System

- Although not required, all variables and constants are declared at the top of main. This practice makes it easier to find their types later on, and it helps the reader know what data is involved in the algorithm.
- For readability, each major step of the algorithm is separated by a blank line and begins with a comment. The class also includes a documentation comment (/**), which you can learn more about in **Appendix B**.

Metric System

- Many algorithms, including the **Convert** program, perform division and modulo together. In both steps, you divide by the same number (**IN_PER_FOOT**).
- When statements including **System.out.printf** get long (generally wider than 80 characters), a common style convention is to break them across multiple lines. The reader should never have to scroll horizontally.

LECTURE 9

The Scanner bug

Scanner Feed-Through Problem

- Now that you've had some experience with Scanner, there is an unexpected behavior we want to warn you about. The following code fragment asks users for their name and age:

```
System.out.print("What is your name? ");
name = in.nextLine();
System.out.print("What is your age? ");
age = in.nextInt();
• System.out.printf("Hello %s, age %d\n", name, age);
```

- The output might look something like this:

```
Hello Grace Hopper, age 45
```

Scanner Feed-Through Problem

- When you read a String followed by an int, everything works just fine. But when you read an int followed by a String, something strange happens.

```
System.out.print("What is your age? ");
age = in.nextInt();
System.out.print("What is your name? ");
name = in.nextLine();
System.out.printf("Hello %s, age %d\n", name, age);
```

- Try running this example code. It doesn't let you input your name, and it immediately displays the output:

```
What is your name? Hello , age 45
```

- To understand what is happening, you need to realize that Scanner doesn't see input as multiple lines like we do. Instead, it gets a "stream of characters" as shown in Figure 3.3

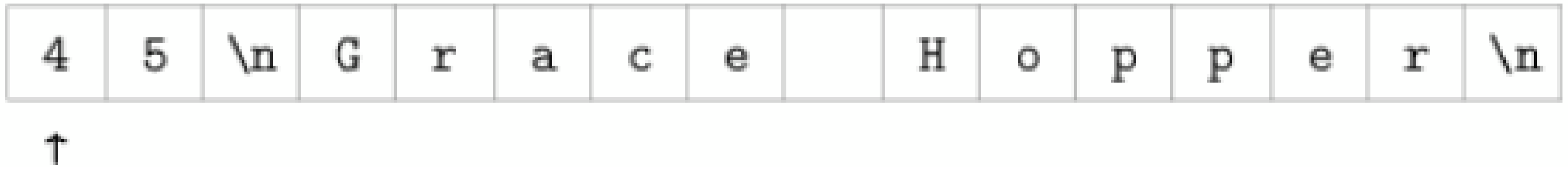


Figure 3.3: A stream of characters as seen by a Scanner

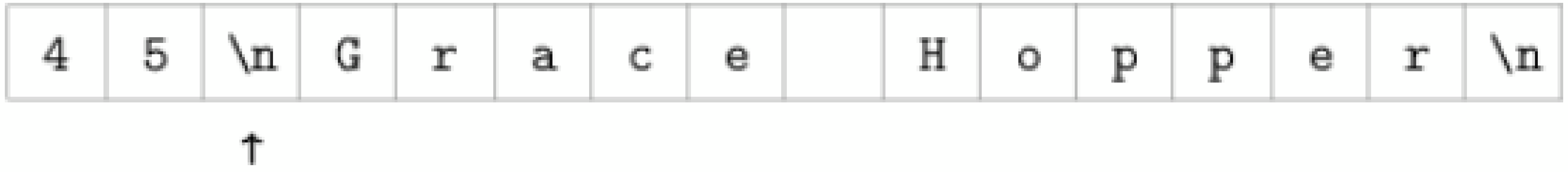


Figure 3.4: A stream of characters after nextInt is called.

Scanner Feed-Through Problem

- At this point, nextInt returns 45. The program then displays the prompt "What is your name? " and calls nextLine, which reads characters until it gets to a newline. But since the next character is already a newline, nextLine returns the empty string "".
- To solve this problem, you need an extra nextLine after nextInt.

```
    System.out.print("What is your age? ");
    age = in.nextInt();
    in.nextLine();  // read the newline
    System.out.print("What is your name? ");
    name = in.nextLine();
    System.out.printf("Hello %s, age %d\n", name, age);
```

- This technique is common when reading int or double values that appear on their own line. First you read the number, and then you read the rest of the line, which is just a newline character.

In-Class Demo Program

- Input three integer
- Find the maximum by Math.math
- Find the maximum by Conditional statements.

MAX.JAVA

Homework

Homework

- Textbook Exercises
- Project 3