Strings, Characters and arithmetic operators

Course: CPSC 1150

Instructor: Dr. Bita Shadgar

Lecture 9

Learning Outcomes

- Use the methods in Math, String and Character class to be able to solve problems in those areas.
- Distinguish between reference type and primitive type
- Distinguish between instance method and static method
- Call and use instance method and static method
- Convert between string and numbers
- Format the output

Using the Java API

- When you encounter a class you aren't familiar with, the API can be a very helpful resource
  - Lists variables and methods belonging to the class
  - Describes what methods do
  - Gives argument and return types of methods
  - Declares the package that class belongs to

Example

Let's look up the Math class and String class in the Java API.

Escape sequence

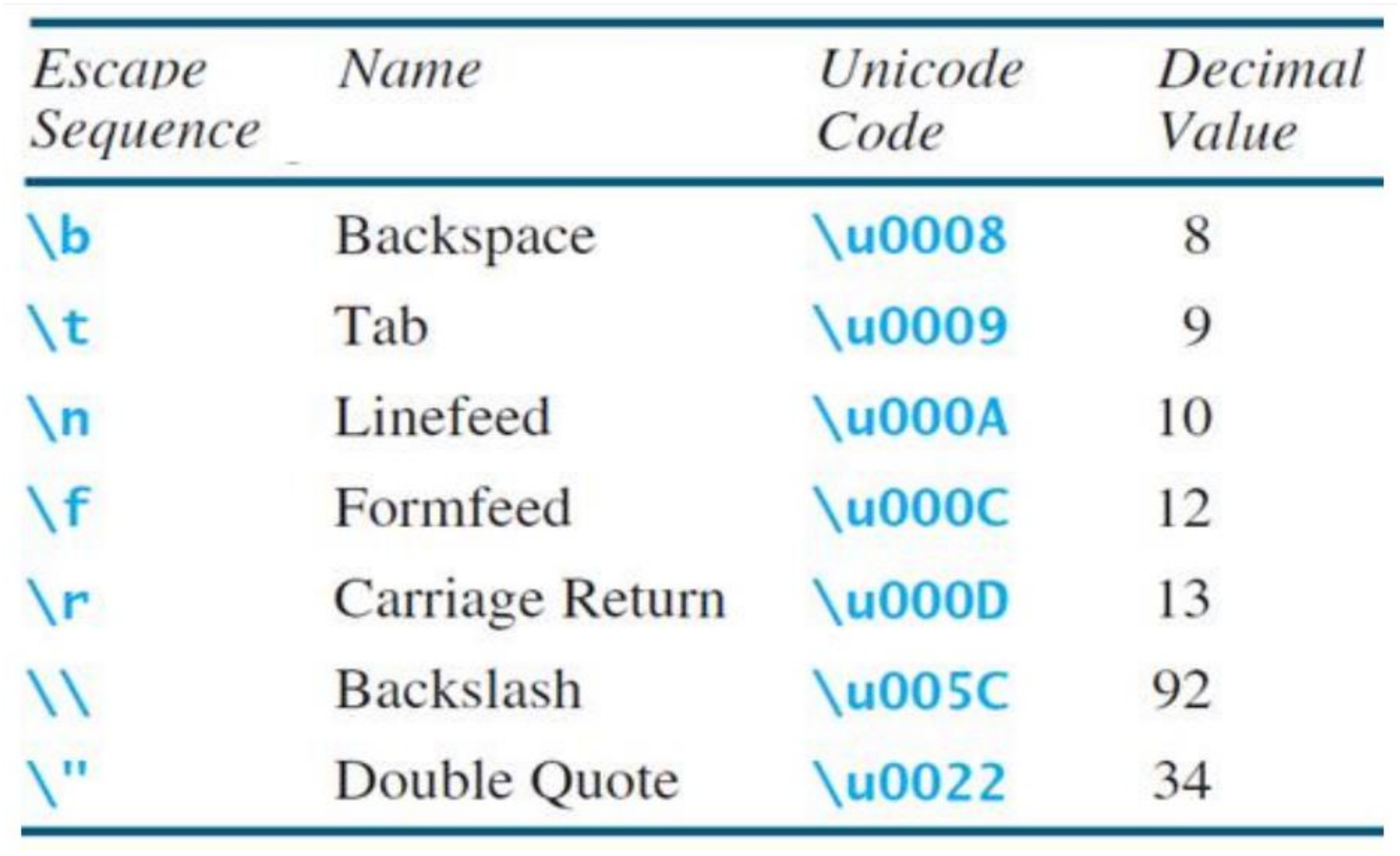
- Some characters either cannot be typed on your keyboard or mean something special to Java
  - Either way, they cannot just be used in a print statement

What happens to the statement…

```
System.out.println("You said "They are Friends" I guess!")
```

- Use an escape sequence to represent those characters
  - A backslash (\) followed by a character or combination of digits
  - The entire escape sequence is interpreted as a single character
- Question: How can we fix the above line of code?

Some useful escape sequence



Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed	\u000A	10
\f	Formfeed	\u000C	12
\r	Carriage Return	\u000D	13
\\	Backslash	\u005C	92
\"	Double Quote	\u0022	34

– **Note:** Any Unicode (i.e., \u004D) is also an escape sequence.

character data type

– Java characters use Unicode takes two bytes, preceded by \u, expressed in four hexadecimal numbers that run from '\u0000' to '\uFFFF'. So, Unicode can represent 65535 + 1 characters.

Example

```
char letter = 'A’;   //(ASCII)
char numChar = '4’;   //(ASCII)
char letter = '\u0041’;   //(Unicode)
System.out.println("\u03b1 \u03b2 \u03b3"); //prints α β γ
```

Casting numeric types into chars

- Any positive integer in the hex range 0x0000 to 0xFFFF can be implicitly cast to a char

```
myChar = 0x0041;    // casting an int into a char, 'A'
```

- Larger integers require explicit casting
  - Only last two bytes are use

```
myChar = (char) 0xD40041;   // also casts into 'A'
```

- Floating-point types can also be explicitly cast in chars
  - Java first casts these into ints

```
myChar = (char) 65.143;    // also casts into 'A'
```

Casting chars into numeric type

- A char literal can be implicitly cast into a numeric type, if the type is large enough to hold the value
- An int and a long are large enough to hold any char
- A short and a byte can only hold some char values

```
byte b1 = 'A’;     // b1 gets 65
int i = '\u00F1’;       // i gets 241
```

- Need explicit casting if the type is not large enough

```
byte b2 = (byte) '\u00F1'; // b2 gets -15
byte b3 = (byte) '\u3AF1'; // b3 gets -15
```

- Question: What is happening in the above two lines of code?

Using numerical operators with chars

- Using numerical operators with chars
- Can use all the comparison operators with char operands
  - Java compares their integer value

Assume ch is a char …

```
if (ch >= 'a' && ch <= 'z’)
      System.out.println(ch + " is a lowercase letter.");
```

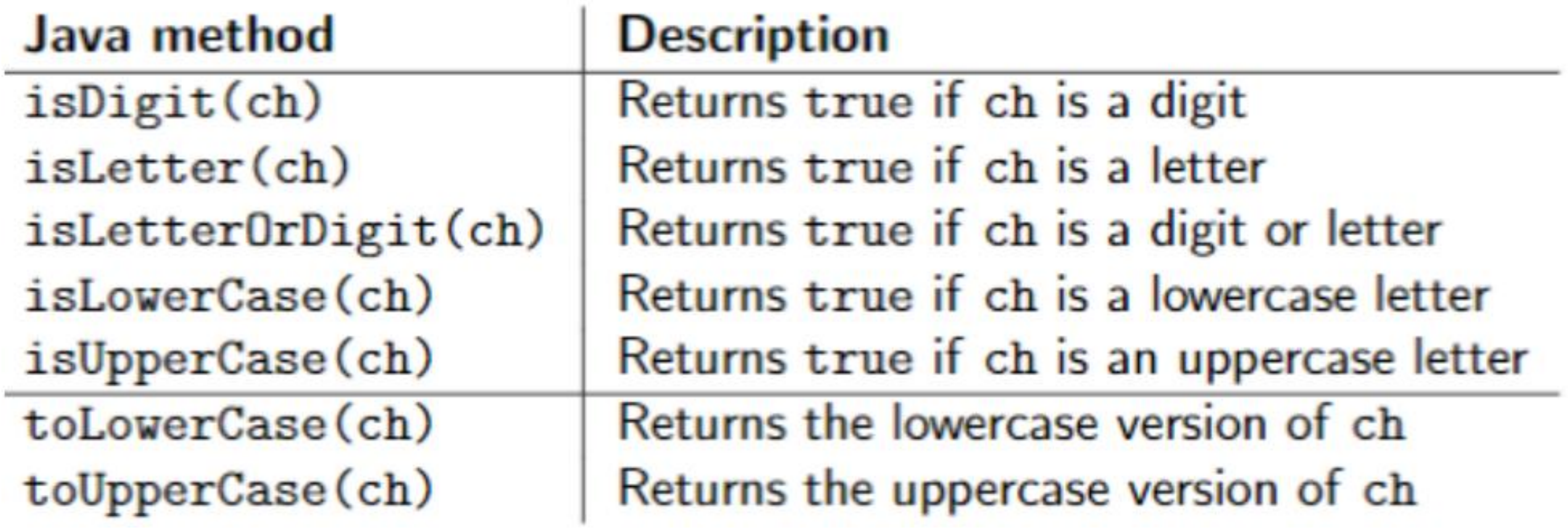
- Can use mathematical operators (+, -, *, /) with char operands
- Operands will be cast into ints

Legal statements

```
int i = 'a' + 'b' + 'c’;      //adds 97 + 98 + 99
System.out.println(i + ", " + (char)i); // 294, Ħ
int j = 'z' - 1;      // 122 - 1
char ch = 'A’;
ch++;   // ch is ’B’
```

Methods in the Character class

– Assume ch is a char



Java method	Description
`isDigit(ch)`	Returns true if ch is a digit
`isLetter(ch)`	Returns true if ch is a letter
`isLetterOrDigit(ch)`	Returns true if ch is a digit or letter
`isLowerCase(ch)`	Returns true if ch is a lowercase letter
`isUpperCase(ch)`	Returns true if ch is an uppercase letter
`toLowerCase(ch)`	Returns the lowercase version of `ch`
`toUpperCase(ch)`	Returns the uppercase version of `ch`

Question: What are the return types of these method?

– Remember that to invoke these methods, you need to type `Character.methodName(ch)`

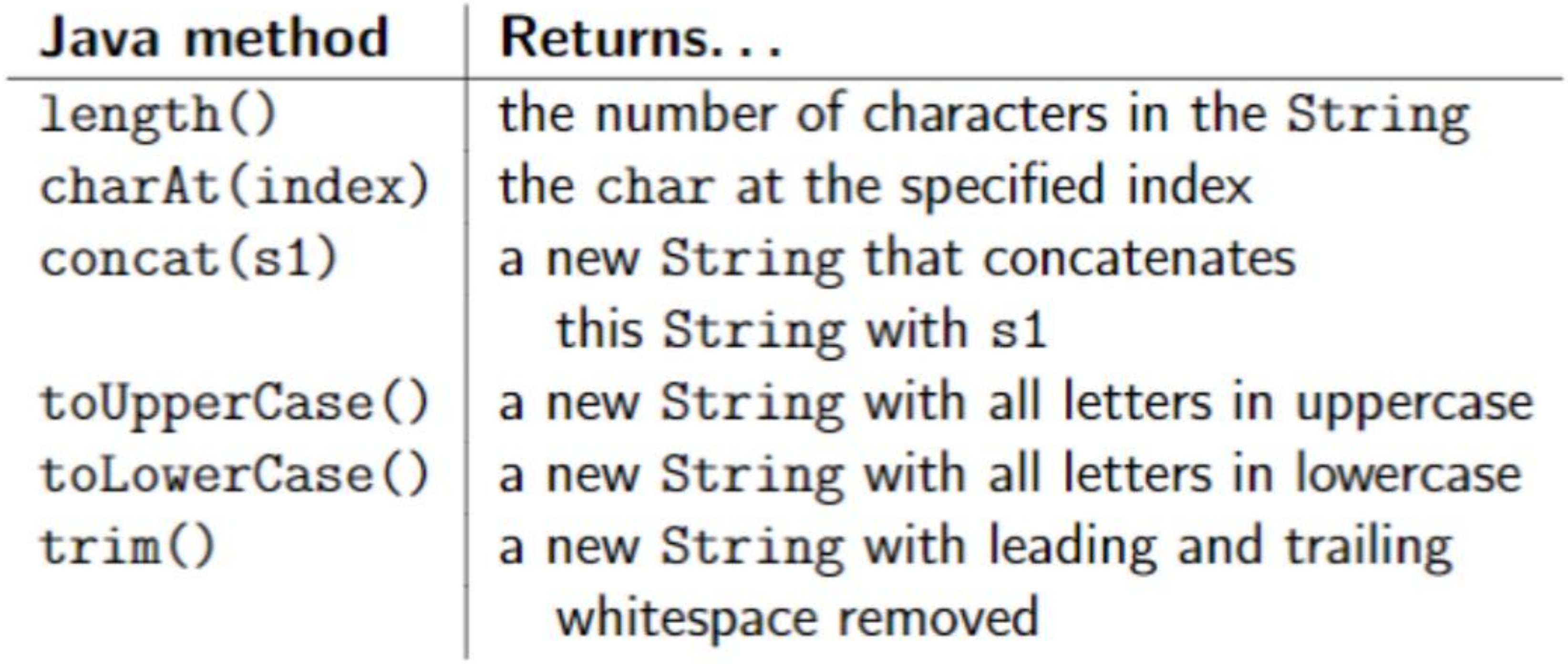
The String type

– Data type to hold a **sequence of characters** (text)
– String literals are enclosed in **double quotes**

Legal statements

```
String courseName = "CPSC 1150";
String studentID = "05994724";  // can create a String
                                // containing escape sequences
String table = "a\tb\tc\n1\t2\t3\n";
```

Some useful String methods



Java method	Returns...
`length()`	the number of characters in the String
`charAt(index)`	the char at the specified index
`concat(s1)`	a new String that concatenates this String with s1
`toUpperCase()`	a new String with all letters in uppercase
`toLowerCase()`	a new String with all letters in lowercase
`trim()`	a new String with leading and trailing whitespace removed

– **Question:** How do you think we can invoke these methods? What’s wrong with typing something like `String.charAt(3)`?

Invoking instance method

- Some methods get invoked using the name of the class, i.e., Math.round(7.634)
  - These are called **static methods**
  - Don't need to have their own object
- Some methods get invoked using the reference variable for a specific instance of the class (an object), i.e., sc.nextShort()
  - These are called **instance methods**
  - Use a particular object that you have created
- The String methods on the previous slide are all instance methods - they all depend on a particular String (an object)

Example
"Happy Birthday".toUpperCase()   returns the String "HAPPY BIRTHDAY"

Reference types vs. primitive type

- Unlike char and numeric data types, String is not a primitive type
- String is known as a reference type
- Any Java class can be used as a reference type for a variable (i.e., Scanner)

```
String courseName = "CPSC 1150";
```

- In the above statement. . .
  - courseName is called a reference variable
  - courseName references a String object
  - The content of the String object is CPSC 115

Practice

Example

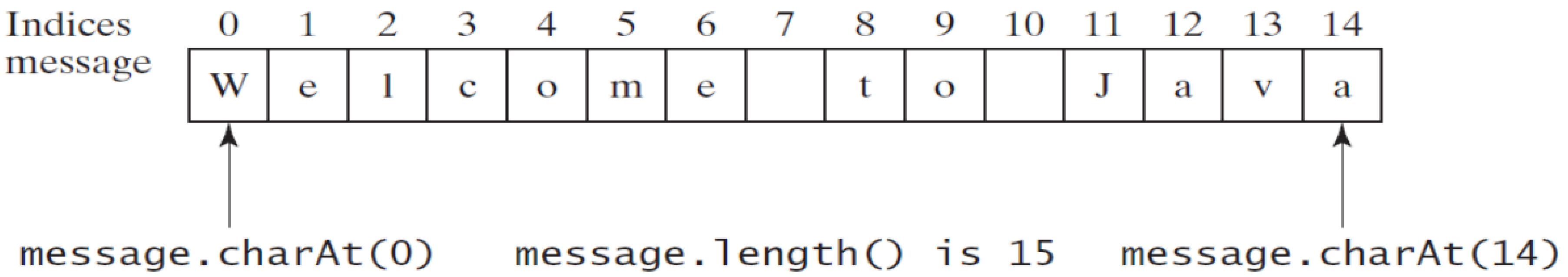
Let's write a program that accepts a String from the user, removes leading and trailing whitespace, and then outputs the number of whitespace characters that were removed.

```java
import java.util.Scanner;
public class Test{
   public static void main(String[] args){
     Scanner input = new Scanner(System.in);
     String data = input.nextLine();
     int be4 = data.length();
     String result = data.trim();
     int after = result.length();
     System.out.println(be4 - after + " character is deleted!");
     input.close();
   }
}
```

Reading from the console

Reading a String from the console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a word: ");
String s1 = input.next();
```

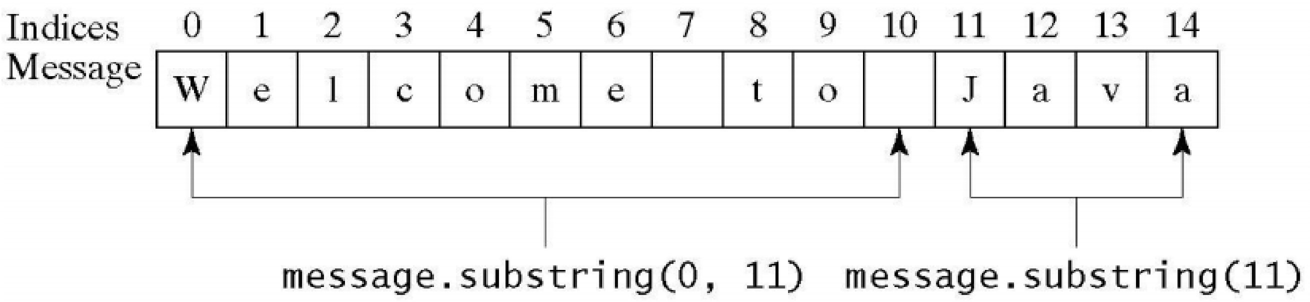


Reading a String from the console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String str = input.nextLine();
char ch = str.charAt(0);
```

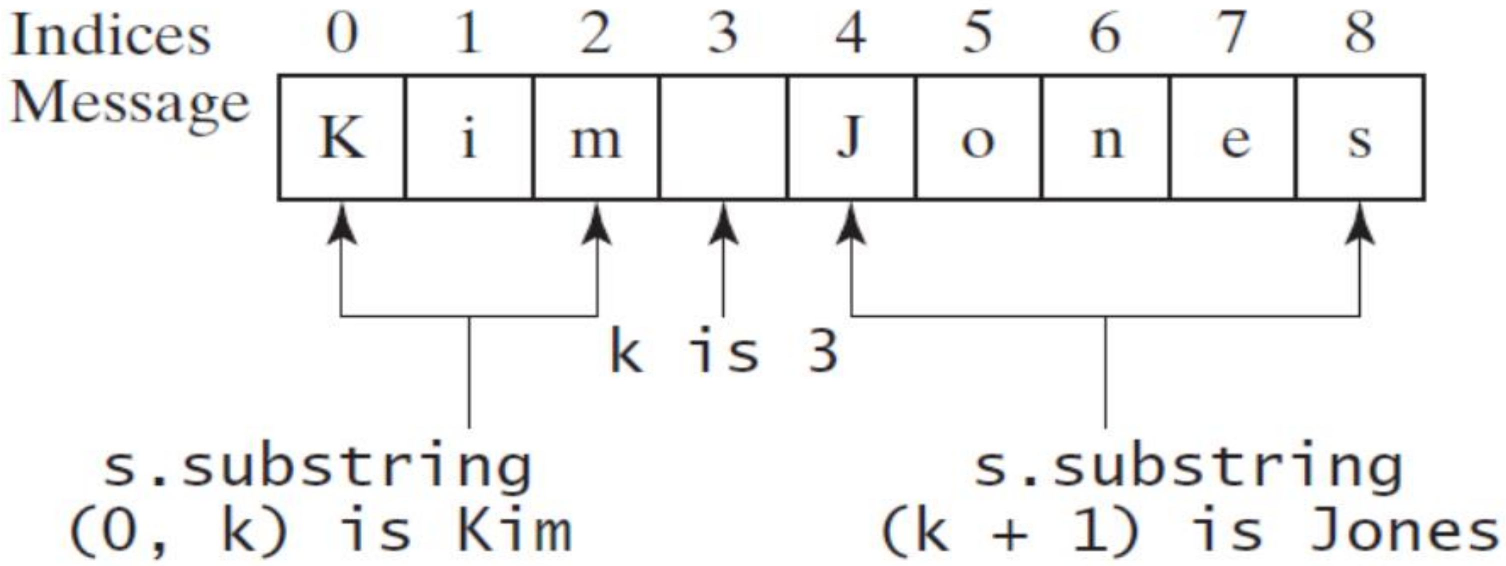
Comparing Strings and obtaining substrings

Method	Description
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than `0`, equal to `0`, or less than `0` to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Note that the character at `endIndex` is not part of the substring.



Example

```
String s = "Kim Jones";
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```



Conversion between Strings and numbers

Convert String to numbers

```
int intValue = Integer.parseInt("1234");
double doubleValue = Double.parseDouble("12.34");
```

– Similar to above examples, other classes like Float, Byte and Short in Java API also provide a method to parse a string into the corresponding number.

Convert numbers to String

```
String s = number + "";
```

Formatting output

Definition and syntax

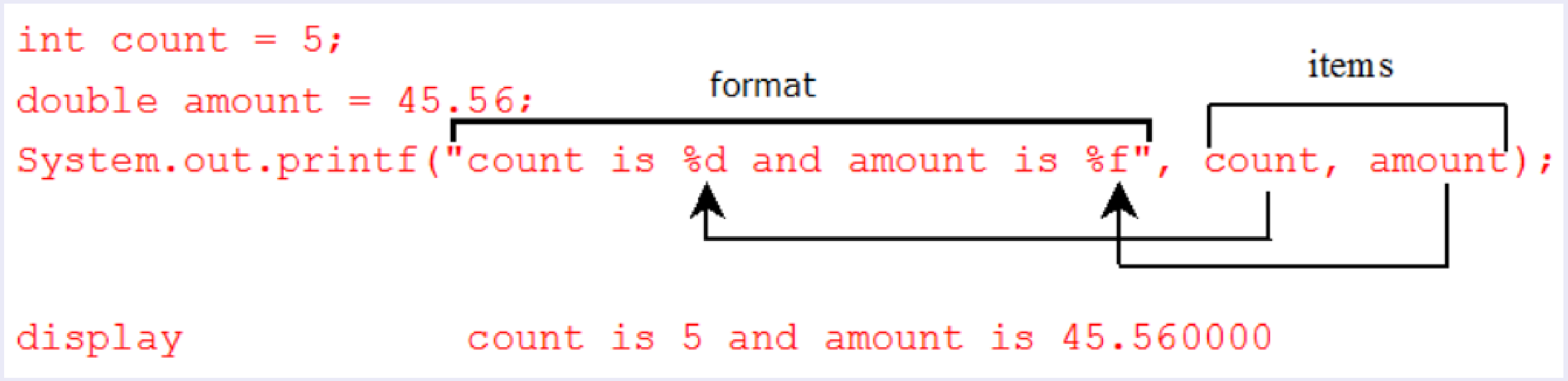
System.out.printf(format, items);   //printf method is used for formatting the output

format is a string that may consist of substrings and format specifiers

item may be a numeric value, character, boolean value, or a string

format specifier specifies how an item should be displayed. It begins with a % sign

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
```

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"

- You can use positive or negative numbers to define the total space for representing the value.
  - Examples : %5d , %20s, %-20c , %10.2f or %-10c
  - Negative number is used for left alignment and positive numbers defines the right alignment.
  - We can also define the number of digits after the decimal point like %10.3f which represents 3 digits after the decimal point

More Practice

- Run each method in the lecture using a simple inputs and make sure you have learnt the task each method is doing!
- Chapter 4 – programming exercises: 4.5, 4.8, 4.9, 4.13, 4.18, 4.21