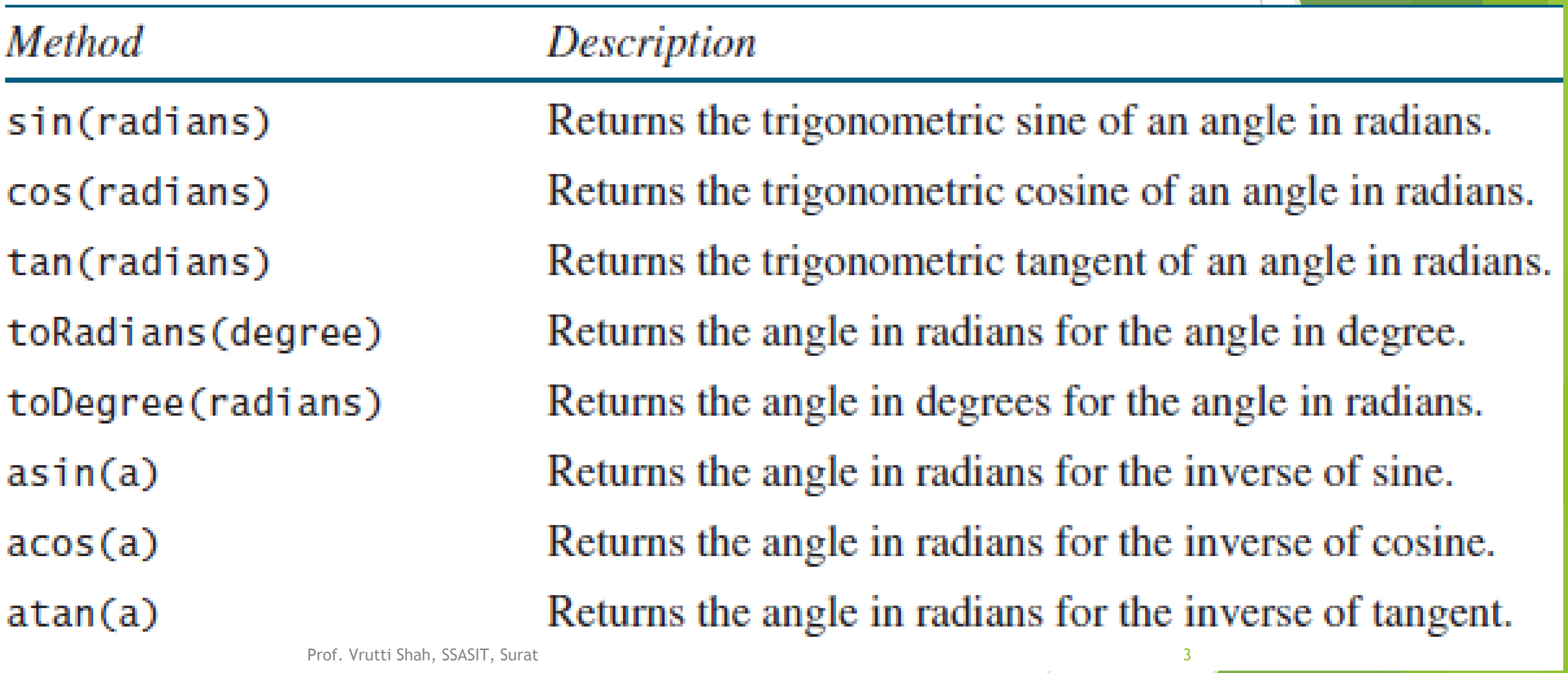
MATHEMATICAL FUNCTIONS, CHARACTERS, AND STRINGS

Common Mathematical Functions

- A method is a group of statements that performs a specific task
- categorized as
  - *trigonometric methods*
  - *exponent methods*
  - *service methods*
- Service methods include the rounding, min, max, absolute, and random methods
- **Math** class provides two useful **double** constants, **PI** and **E**
  - **Math.PI**
  - **Math.E**

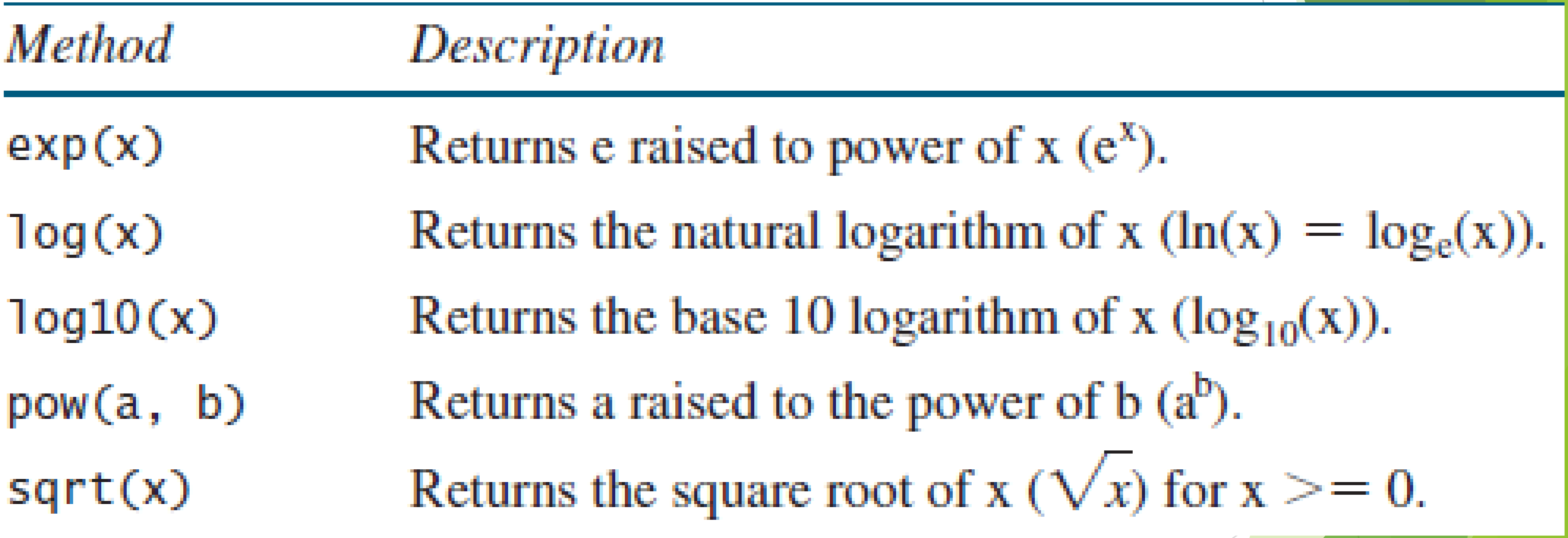
Trigonometric Methods (Math class)



Method	*Description*
`sin(radians)`	Returns the trigonometric sine of an angle in radians.
`cos(radians)`	Returns the trigonometric cosine of an angle in radians.
`tan(radians)`	Returns the trigonometric tangent of an angle in radians.
`toRadians(degree)`	Returns the angle in radians for the angle in degree.
`toDegree(radians)`	Returns the angle in degrees for the angle in radians.
`asin(a)`	Returns the angle in radians for the inverse of sine.
`acos(a)`	Returns the angle in radians for the inverse of cosine.
`atan(a)`	Returns the angle in radians for the inverse of tangent.

- parameter for sin, cos, and tan is an angle in radians
- **Rad = Deg** × π/180
- return value for asin, acos, and atan is a degree in radians
  - range between -π/2 and π/2
- Math.toDegrees(Math.PI / 2) returns 90.0
- Math.toRadians(30) returns 0.5236 (same as π/6)
- Math.sin(0) returns 0.0
- Math.sin(Math.toRadians(270)) returns -1.0
- Math.sin(Math.PI / 6) returns 0.5
- Math.sin(Math.PI / 2) returns 1.0
- Math.cos(0) returns 1.0
- Math.cos(Math.PI / 6) returns 0.866
- Math.cos(Math.PI / 2) returns 0
- Math.asin(0.5) returns 0.523598333 (same as π/6)
- Math.acos(0.5) returns 1.0472 (same as π/3)
- Math.atan(1.0) returns 0.785398 (same as π/4)

Exponent Methods



Method	*Description*
`exp(x)`	Returns e raised to power of x (e^x).
`log(x)`	Returns the natural logarithm of x ($\ln(x) = \log_e(x)$).
`log10(x)`	Returns the base 10 logarithm of x ($\log_{10}(x)$).
`pow(a, b)`	Returns a raised to the power of b (a^b).
`sqrt(x)`	Returns the square root of x ($\sqrt{x}$) for $x >= 0$.

Exponent Methods (Contd…)

- Math.exp(1) returns 2.71828
- Math.log(Math.E) returns 1.0
- Math.log10(10) returns 1.0
- Math.pow(2, 3) returns 8.0
- Math.pow(3, 2) returns 9.0
- Math.pow(4.5, 2.5) returns 22.91765
- Math.sqrt(4) returns 2.0
- Math.sqrt(10.5) returns 4.24

Rounding Methods

Method	Description
ceil(x)	x is rounded up to its nearest integer. This integer is returned as a double value
floor(x)	x is rounded down to its nearest integer. This integer is returned as a double value.
rint(x)	x is rounded up to its nearest integer. If x is equally close to two integers, the even one is returned as a double value.
round(x)	Returns (int)Math.floor(x + 0.5) if x is a float and returns (long)Math.floor(x + 0.5) if x is a double.

Rounding Methods (Contd…)

- Math.ceil(2.1) returns 3.0
- Math.ceil(2.0) returns 2.0
- Math.ceil(-2.0) returns -2.0
- Math.ceil(-2.1) returns -2.0
- Math.floor(2.1) returns 2.0
- Math.floor(2.0) returns 2.0
- Math.floor(-2.0) returns -2.0
- Math.floor(-2.1) returns -3.0
- Math.rint(2.1) returns 2.0

Rounding Methods (Contd…)

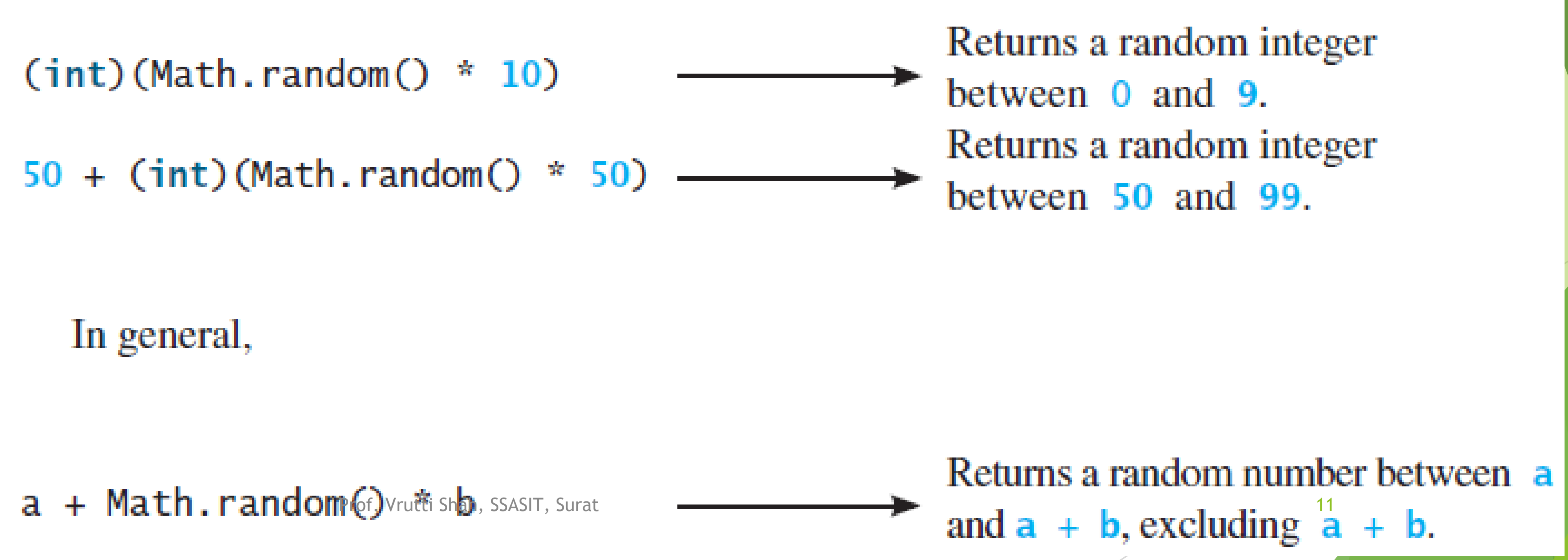
- Math.rint(-2.0) returns -2.0
- Math.rint(-2.1) returns -2.0
- Math.rint(2.5) returns 2.0
- Math.rint(4.5) returns 4.0
- Math.rint(-2.5) returns -2.0
- Math.round(2.6f) returns 3 // Returns int
- Math.round(2.0) returns 2 // Returns long
- Math.round(-2.0f) returns -2 // Returns int
- Math.round(-2.6) returns -3 // Returns long
- Math.round(-2.4) returns -2 // Returns long

min, max, and abs Methods

- **min** and **max** methods return the minimum and maximum numbers of two numbers (**int**, **long**, **float**, or **double**)
- **abs** method returns the absolute value of the number (**int**, **long**, **float**, or **double**)
- Math.max(2, 3) returns 3
- Math.max(2.5, 3) returns 3.0
- Math.min(2.5, 4.6) returns 2.5
- Math.abs(-2) returns 2
- Math.abs(-2.1) returns 2.1

random Method

- method generates a random **double** value ≥ 0.0 and < 1.0
- **0 <= Math.random() < 1.0**



```
(int)(Math.random() * 10)
```
→ Returns a random integer between 0 and 9.

```
50 + (int)(Math.random() * 50)
```
→ Returns a random integer between 50 and 99.

In general,

```
a + Math.random() * b
```
→ Returns a random number between a and a + b, excluding a + b.

Character Data Type and Operations

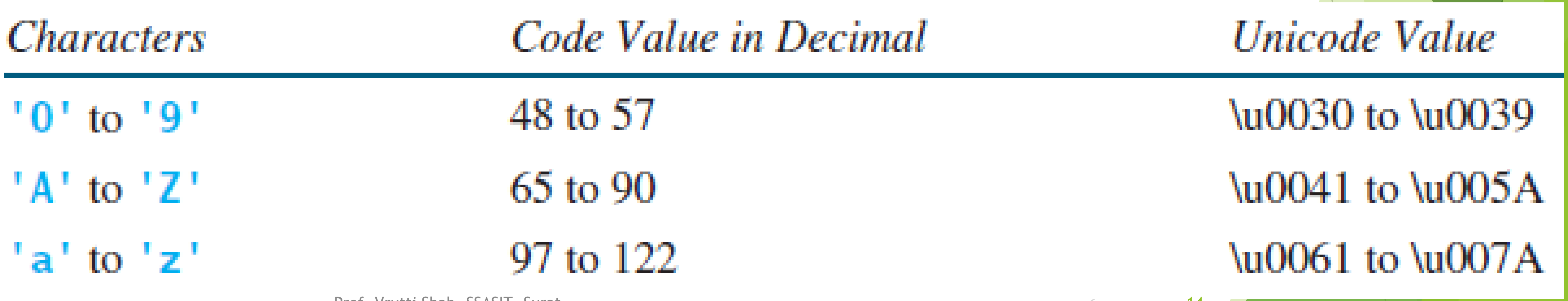
- *represents a single character*
- enclosed in single quotation marks
- `char letter = 'A';`

Unicode and ASCII code

- Mapping a character to its binary representation is called *encoding*
- How characters are encoded is defined by an *encoding scheme*.
- Java supports *Unicode*, 16-bit character encoding
- Unicode standard has been extended to allow up to 1,112,064 characters
- Characters that go beyond the original 16-bit limit are called *supplementary characters*
- 16-bit Unicode
  - takes two bytes, preceded by **\u**
  - expressed in four hexadecimal digits that run from **\u0000** to **\uFFFF**

Unicode and ASCII code (Contd…)

- *ASCII* is an 8-bit encoding scheme
- represents all uppercase and lowercase letters, digits, punctuation marks, and control characters
- Unicode includes ASCII code, with \u0000 to \u007F corresponding to the 128 ASCII characters



Characters	*Code Value in Decimal*	*Unicode Value*
'0' to '9'	48 to 57	\u0030 to \u0039
'A' to 'Z'	65 to 90	\u0041 to \u005A
'a' to 'z'	97 to 122	\u0061 to \u007A

Unicode and ASCII code (Contd…)

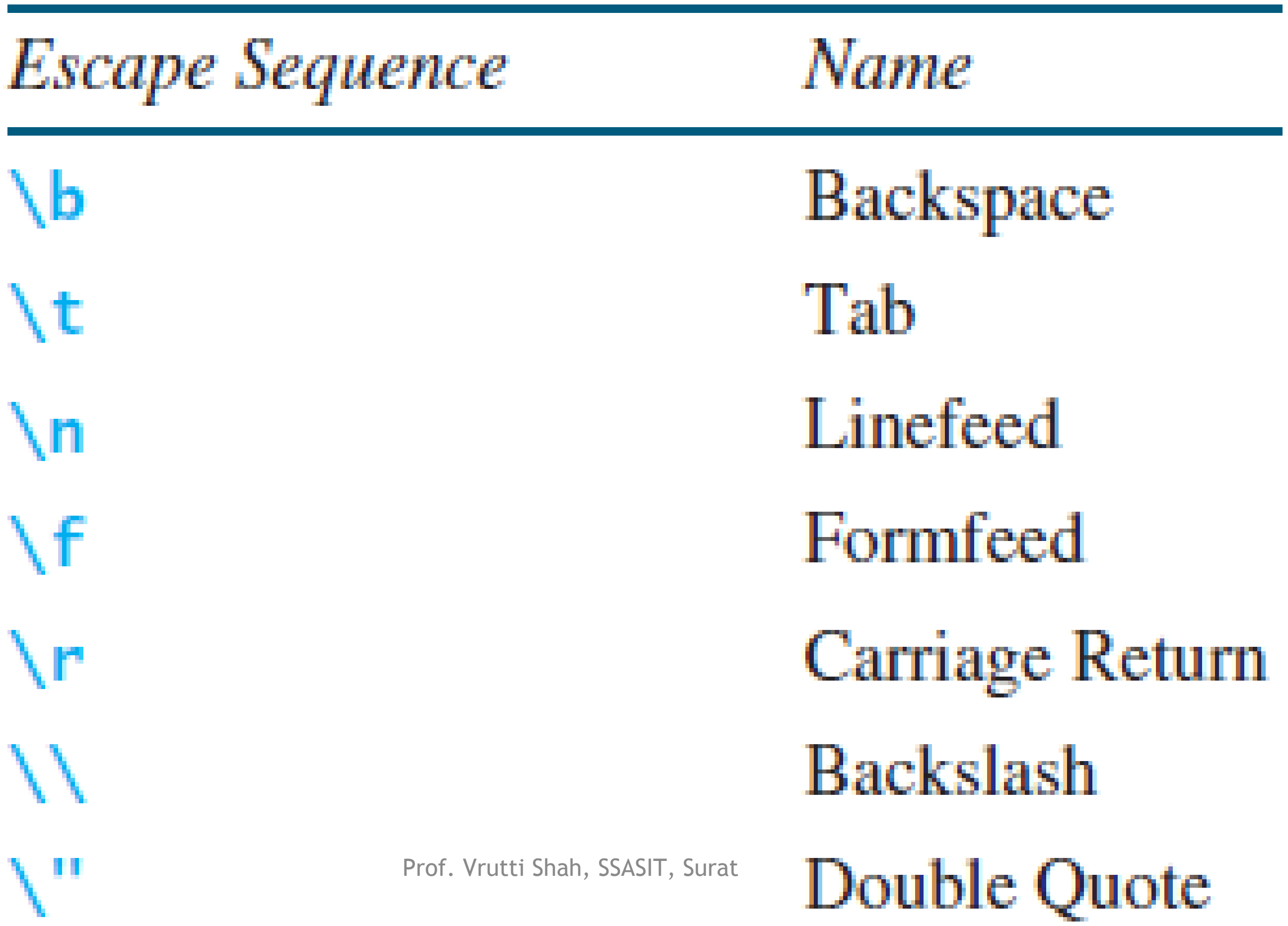
- char letter = 'A';
- char letter = '\u0041';
- increment and decrement operators can be used on char variables to get next or preceding Unicode character.
- char ch = 'a';

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

Escape Sequences for Special Characters

- System.out.println(**"He said "Java is fun""**);   //output??
- *escape sequence*, consists of a backslash (\) followed by a character or a combination of digits
- **\t** is an escape sequence for the Tab character
- **\u03b1** is used to represent a Unicode

Escape Sequences for Special Characters (Contd…)



Escape Sequence	*Name*
`\b`	Backspace
`\t`	Tab
`\n`	Linefeed
`\f`	Formfeed
`\r`	Carriage Return
`\\`	Backslash
`\"`	Double Quote

Casting between **char** and Numeric Types

- char can be cast into any numeric type, and vice versa
- When an integer is cast into a char, only its lower 16 bits of data are used; the other part is ignored
- char ch = (char)**0XAB0041**;
- When a floating-point value is cast into a **char**, the floating-point value is first cast into an **int**, which is then cast into a **char**
- char ch = (char)**65.25**;
- When a **char** is cast into a numeric type, the character's Unicode is cast into the specified numeric type
- int i = (int)'A';

Casting between **char** and Numeric Types (Contd…)

- **byte** b = '**a**';
- **int** i = '**a**';
- following casting is incorrect, because the Unicode **\uFFF4** cannot fit into a byte
  - byte b = '\uFFF4';
- byte b = (byte)'\uFFF4';
- All numeric operators can be applied to **char** operands
- char operand is automatically cast into a number if the other operand is a number or a character
- If the other operand is a string, the character is concatenated with the string

- **int** i = **'2'** + **'3'**; // (int)'2' is 50 and (int)'3' is 51

  System.out.println(**"i is "** + i);

  int j = **2** + **'a'**; // (int)'a' is 97

  System.out.println(**"j is "** + j);

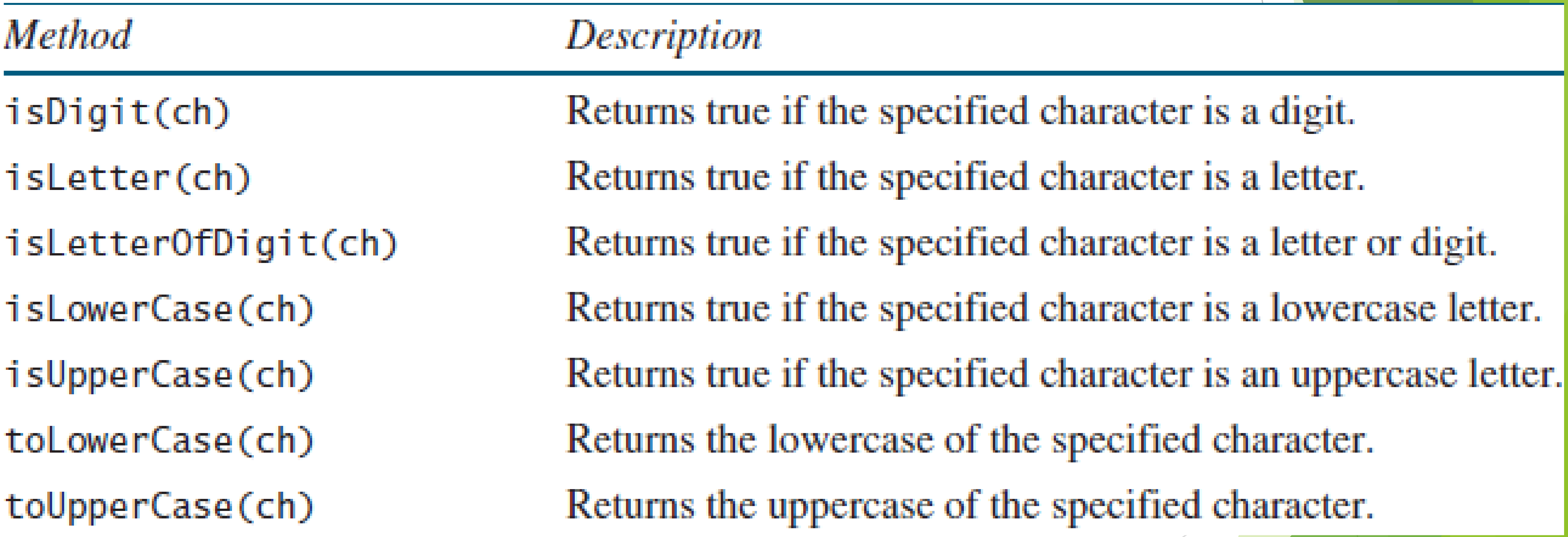
  System.out.println(j + **" is the Unicode for character "**+ (**char**)j);

  System.out.println(**"Chapter "** + **'2'**);

Comparing and Testing Characters

- Two characters can be compared using the relational operators
- **'a' < 'b'** is true
- **'a' < 'A'** is false
- **'1' < '8'** is true

methods in the **Character** class

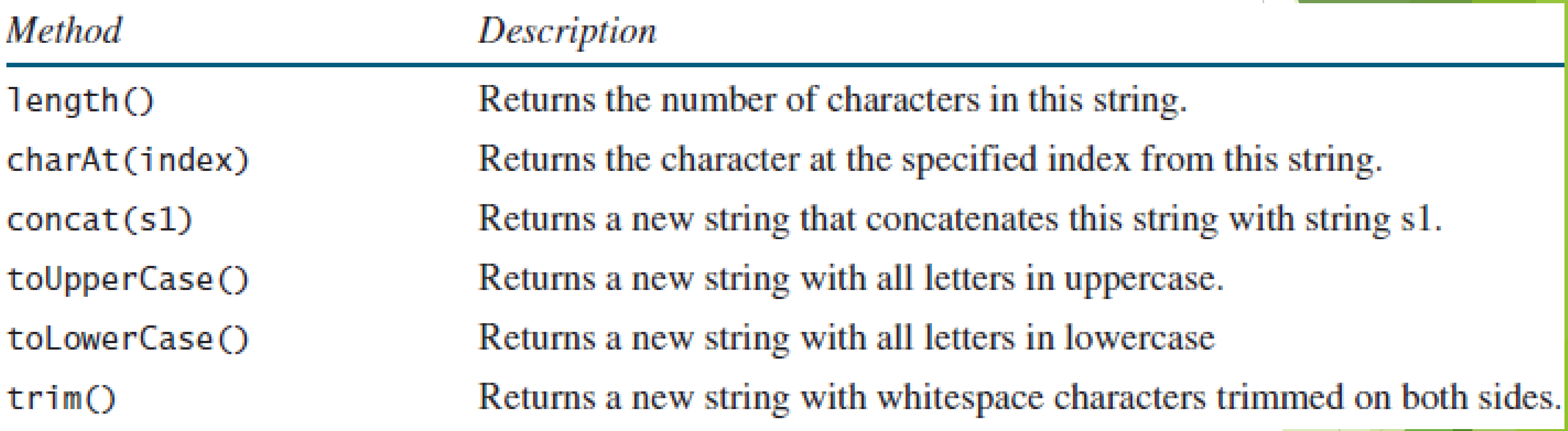


Method	*Description*
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOfDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

The String Type

- *A string is a sequence of characters*
- To represent a string of characters, use the data type called **String**
- String message = **"Welcome to Java"**;
- **String** is a predefined class in the Java library
- **String** type is not a primitive type. It is known as a *reference type*
- variable declared by a reference type is known as a reference variable that references an object

String methods



Method	*Description*
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

String methods (Contd…)

- The methods of String can only be invoked from a specific string instance
- these methods are called *instance methods*
- A non-instance method is called a *static method*.
- static method can be invoked without using an object
- All the methods defined in the **Math** class are static methods
- syntax to invoke an instance method is
  - **reference-Variable.methodName(arguments)**
- syntax to invoke a static method is
  - **ClassName.methodName(arguments)**
- **"Welcome to Java".length() is allowed in Java**

Reading a String from the Console

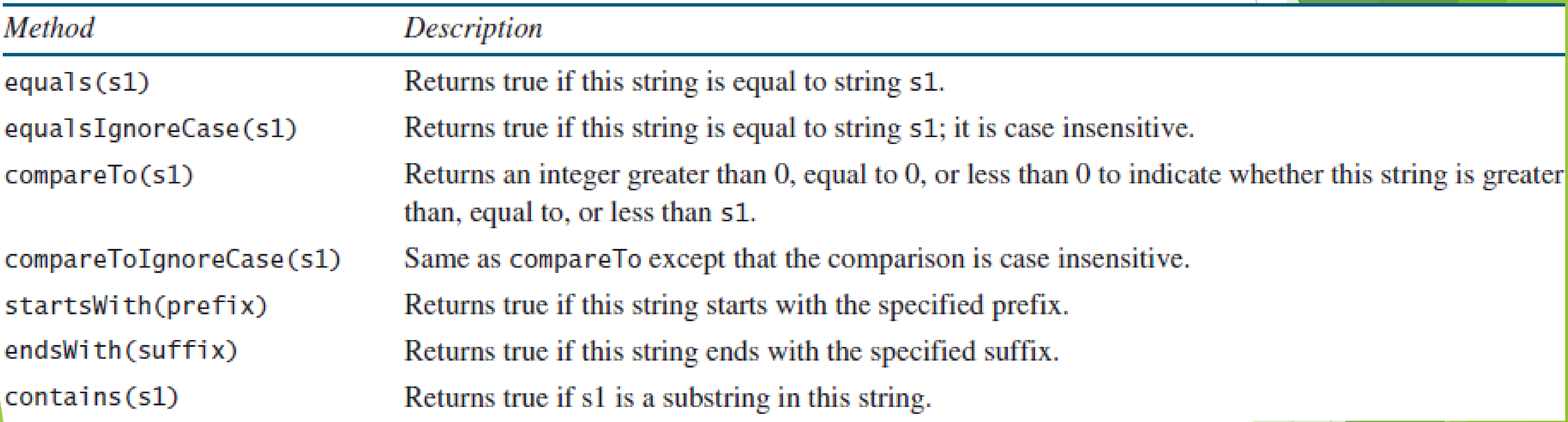
- Scanner input = **new** Scanner(System.in);
- String s1 = input.next();
- next() method reads a string that ends with a whitespace character
- If input is “Welcome to Java” then s1=??
- nextLine() method to read an entire line of text
  - reads a string that ends with the *Enter* key pressed
- To avoid input errors, do not use nextLine() after nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or next()

Reading a Character from the Console

- use the **nextLine()** method to read a string and then invoke the **charAt(0)** method on the string to return a character
- Scanner input = **new** Scanner(System.in);

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

Comparing Strings



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.
`contains(s1)`	Returns true if s1 is a substring in this string.

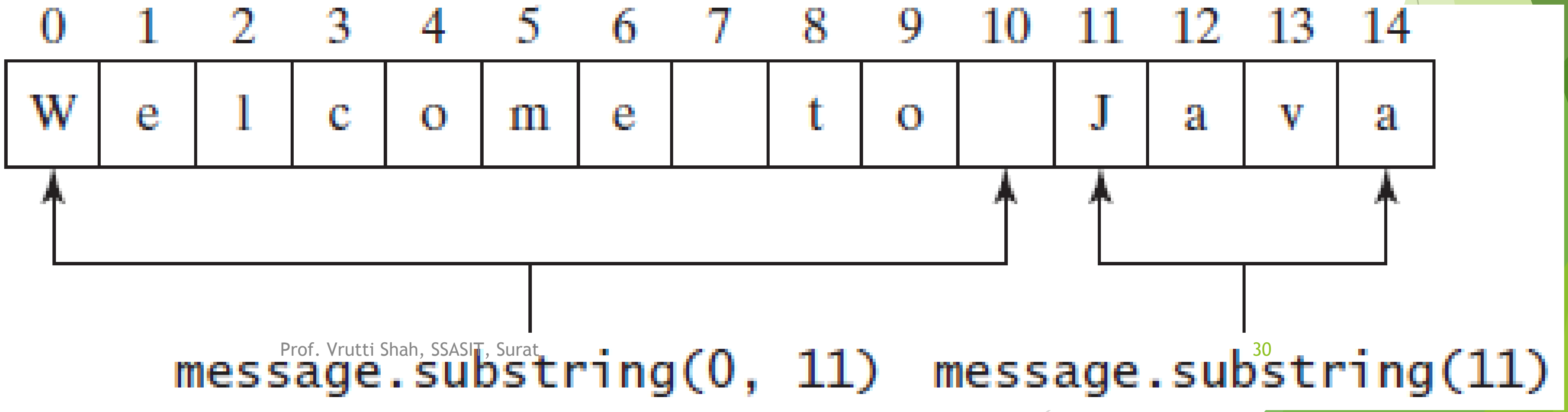
Comparing Strings (Contd…)

- **==** operator checks only whether **string1** and **string2** refer to the same object
- it does not tell you whether they have the same contents
- **compareTo** method can also be used to compare two strings
- suppose **s1** is **abc** and **s2** is **abg**, **s1.compareTo(s2)** returns **-4**.
- **"Welcome to Java".startsWith("We")** returns **true**.
- **"Welcome to Java".startsWith("we")** returns **false**.
- **"Welcome to Java".endsWith("va")** returns **true**.
- **"Welcome to Java".contains("to")** returns **true**.
- **"Welcome to Java".contains("To")** returns **false**.

Obtaining Substrings

- obtain a single character from a string using the **charAt** method
- obtain a substring from a string using the **substring** method in the **String** class

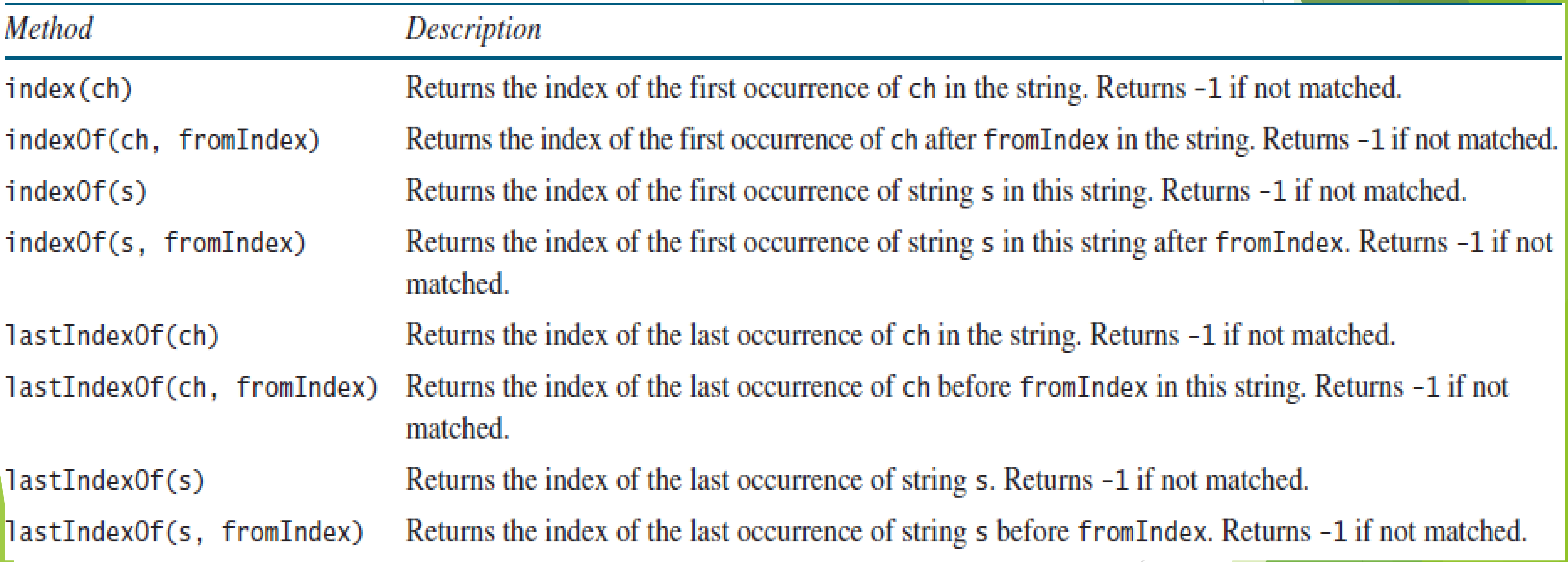
Method	*Description*
substring(beginIndex)	Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string
substring(beginIndex, endIndex)	Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex – 1,



Obtaining Substrings (Contd…)

- If **beginIndex** is **endIndex**, **substring(beginIndex, endIndex)** returns an empty string with length **0**.
- If **beginIndex** > **endIndex**, it would be a runtime error.

Finding a Character or a Substring in a String



Method	*Description*
`index(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

Finding a Character or a Substring in a String (Contd…)

- "Welcome to Java".indexOf('W') returns 0.
- "Welcome to Java".indexOf('o') returns 4.
- "Welcome to Java".indexOf('o', 5) returns 9.
- "Welcome to Java".indexOf("come") returns 3.
- "Welcome to Java".indexOf("Java", 5) returns 11.
- "Welcome to Java".indexOf("java", 5) returns -1.
- "Welcome to Java".lastIndexOf('W') returns 0.
- "Welcome to Java".lastIndexOf('o') returns 9.
- "Welcome to Java".lastIndexOf('o', 5) returns 4.
- "Welcome to Java".lastIndexOf("come") returns 3.
- "Welcome to Java".lastIndexOf("Java", 5) returns -1.
- "Welcome to Java".lastIndexOf("Java") returns 11.

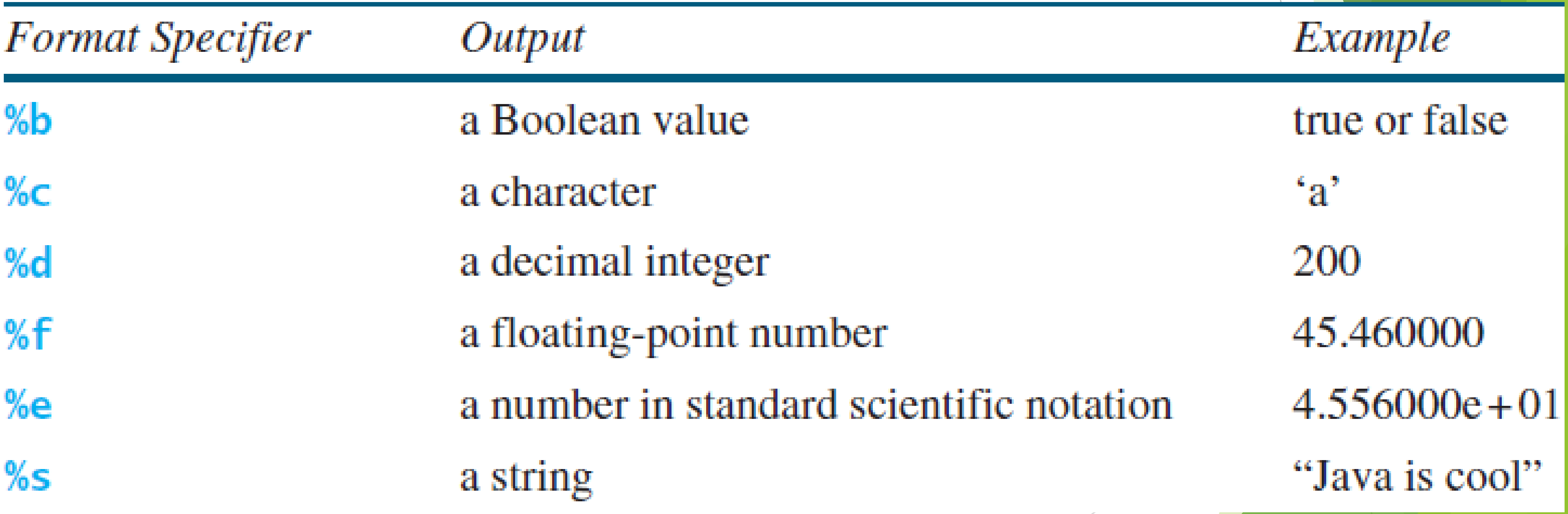
Conversion between Strings and Numbers

- To convert a string into an **int** value, use the **Integer.parseInt** method
  - **int** intValue = Integer.parseInt(intString);
- convert a string into a **double** value, use the **Double.parseDouble** method
  - **double** doubleValue = Double.parseDouble(doubleString);
- If string is not a numeric string, conversion would cause a runtime error
- convert a number into a string
  - String s = number + "";

- String message = **"Welcome " + "to " + "Java"**;
- String s = **"Chapter"** + **2**; // s becomes Chapter2
- String s1 = **"Supplement"** + **'B'**; // s1 becomes SupplementB
- message += **" and Java is fun"**;
- **"Welcome".toLowerCase()** returns a new string **welcome**.
- **"Welcome".toUpperCase()** returns a new string **WELCOME**.

Formatting Console Output

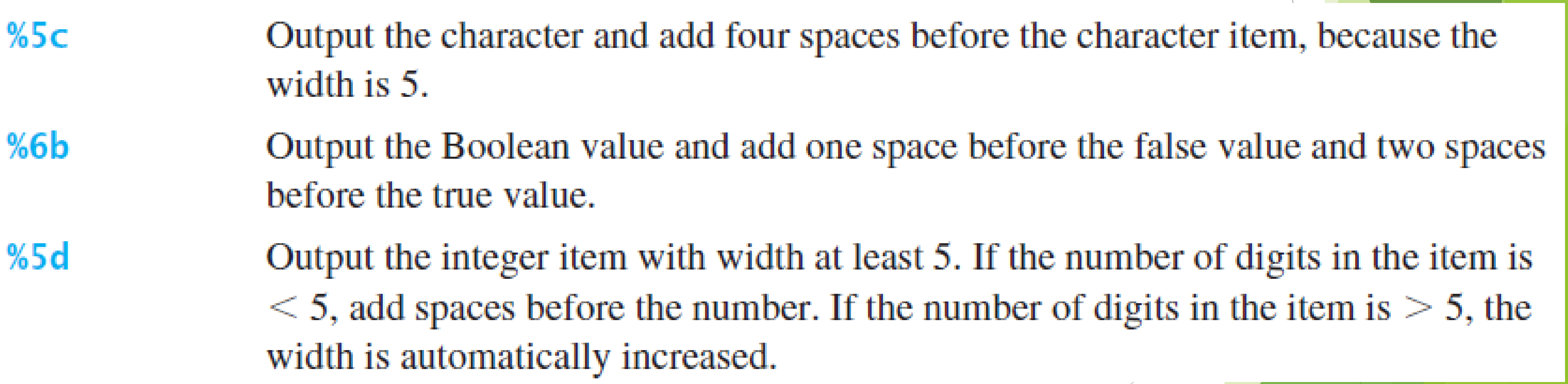
- System.out.printf(format, item1, item2, ..., item*k*)
- format is a string that may consist of substrings and format specifiers
- format specifier consists of a percent sign (%) followed by a conversion code



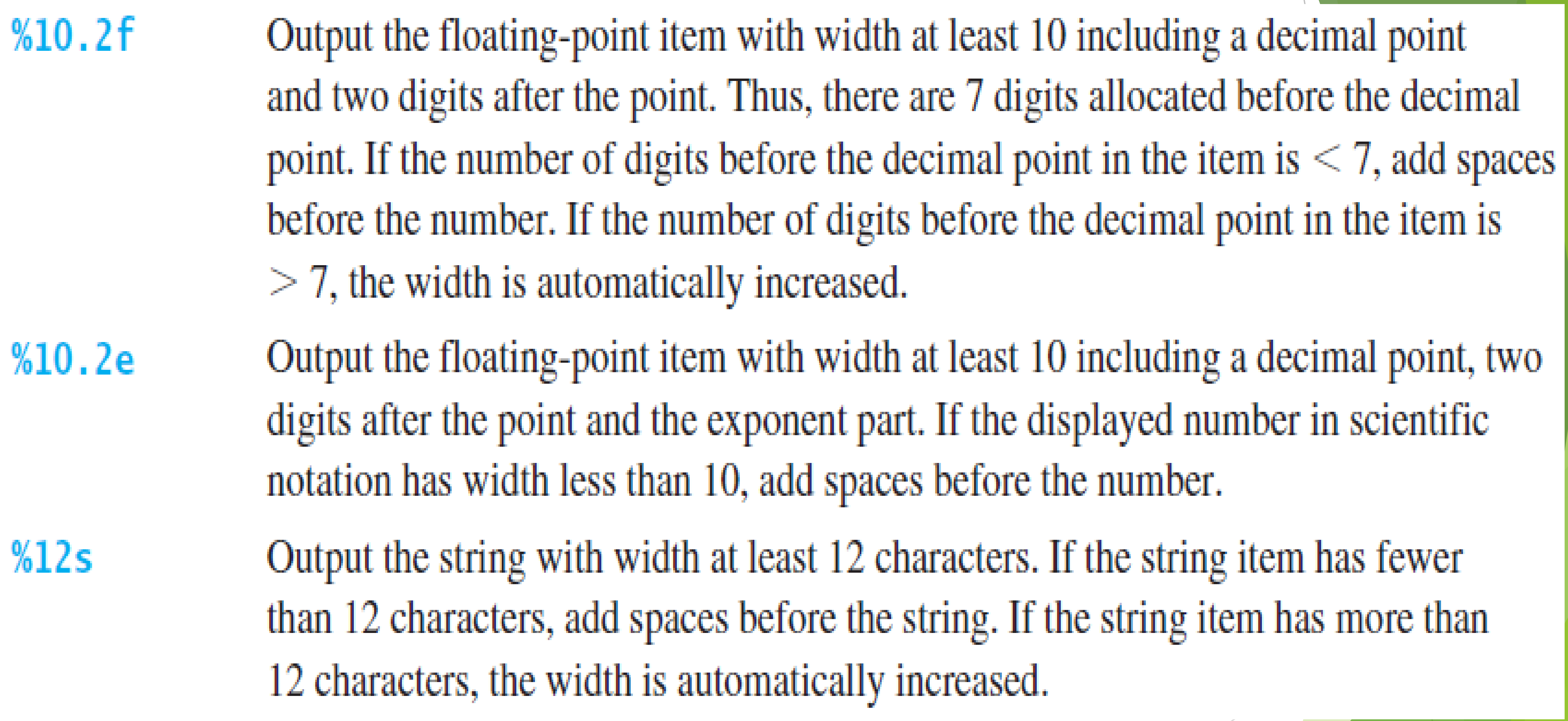
Format Specifier	*Output*	*Example*
%b	a Boolean value	true or false
%c	a character	‘a’
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s	a string	“Java is cool”

Formatting Console Output (Contd…)

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



%5c	Output the character and add four spaces before the character item, because the width is 5.
%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. If the number of digits in the item is $>$ 5, the width is automatically increased.



`%10.2f`	Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus, there are 7 digits allocated before the decimal point. If the number of digits before the decimal point in the item is < 7, add spaces before the number. If the number of digits before the decimal point in the item is > 7, the width is automatically increased.
`%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 characters. If the string item has fewer than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased.

- System.out.printf("%8d%8s%8.1f\n", 1234, "Java", 5.63);
- System.out.printf("%-8d%-8s%-8.1f \n", 1234, "Java", 5.63);
- System.out.printf("amount is %f %e\n", 32.32, 32.32);
- System.out.printf("amount is %5.2%% %5.4e\n", 32.327, 32.32);
- System.out.printf("%6b\n", (1 > 2));
- System.out.printf("%6s\n", "Java");
- System.out.printf("%-6b%s\n", (1 > 2), "Java");
- System.out.printf("%6b%-8s\n", (1 > 2), "Java");