

# Chapter 4

## Mathematical Functions, Characters, and Strings

# Outline

1. Java Class Library
2. Class *Math*
3. Character Data Type
4. Class *String*
5. `printf` Statement

# 1. Java Class Library

- A *class library* is a collection of classes that we use when developing programs
- The *Java standard class library* is part of any Java development environment
- The library classes are not part of the Java language per se, but we rely on them heavily
- Various library classes we've already used in our programs, such as `System`, `Scanner`, and `Random`
- Other class libraries can be obtained through third party vendors, or you can create them yourself
- Classes **must be imported** into the program

# Packages

- The classes of the Java standard class library are organized into *packages*
- Sample packages in the standard class library are:

<u>Package</u>	<u>Purpose</u>
java.lang	General support ( <i>Character, Math, System, String, Number, ...</i> )
java.util	Utilities ( <i>Scanner, Date, Random, Calendar, ...</i> )
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities
java.net	Network communication
javax.xml.parsers	XML document processing

# import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Scanner
```

- Or you can *import* the class, and then use just the class name

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

- To import all classes in a particular package, you can use the *\* wildcard character*

```
import java.util.*; // wildcard
```

# import Declaration

- All classes of the `java.lang` package are imported automatically into all programs
- It's as if all programs contain the following line:

```
import java.lang.*;
```

- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs
- The `Scanner` class, on the other hand, is part of the `java.util` package, and therefore must be imported

## 2. Class *Math*

- The **Math class** is part of the **java.lang** package
- The Math class contains methods (called **class methods**) that perform various mathematical functions:
  - PI constant
  - E (base of natural logarithms) constant
  - Trigonometric Methods
  - Exponent Methods
  - Rounding Methods
  - min, max, abs, and random Methods
- Methods in the Math class are called **static methods**
- **Static methods can be invoked through the class name**  
– no object of the Math class is needed

```
Double value = Math.cos(90) + Math.sqrt(delta);
```

# Example

```
import java.util.Scanner;
public class Quadratic
{
    public static void main (String[] args)
    {
        int a, b, c;  //  $ax^2 + bx + c$ 
        double discriminant, root1, root2;
        Scanner scan = new Scanner (System.in);

        System.out.print ("Enter the coefficient of x squared: ");
        a = scan.nextInt();
        System.out.print ("Enter the coefficient of x: ");
        b = scan.nextInt();
        System.out.print ("Enter the constant: ");
        c = scan.nextInt();

        // Use quadratic formula to compute the roots.

        discriminant = Math.pow(b, 2) - (4 * a * c);
        root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
        root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);

        System.out.println ("Root #1: " + root1);
        System.out.println ("Root #2: " + root2);
    }
}
```



# Example

## Output:

Enter the coefficient of x squared: 3

Enter the coefficient of x: 8

Enter the constant: 4

Root #1: -0.6666666666666666

Root #2: -2.0

Enter the coefficient of x squared: 2

Enter the coefficient of x: 4

Enter the constant: 8

Root #1: NaN

Root #2: NaN

NaN indicates **undefined root** due to square root of negative value (sqrt of  $b^2-4ac$ )

# Trigonometric Methods

- `sin(double a)`
- `cos(double a)`
- `tan(double a)`
- `acos(double a)`
- `asin(double a)`
- `atan(double a)`

## Examples:

`Math.sin(0)` returns 0.0

`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/2)` returns 0

`Math.cos(Math.PI/6)` returns 0.866

# Exponent Methods

- **`exp(double a)`**  
Returns  $e$  raised to the power of  $a$ .
- **`log(double a)`**  
Returns the natural logarithm of  $a$ .
- **`log10(double a)`**  
Returns the 10-based logarithm of  $a$ .
- **`pow(double a, double b)`**  
Returns  $a$  raised to the power of  $b$ .
- **`sqrt(double a)`**  
Returns the square root of  $a$ .

## Examples:

**`Math.exp(1)` returns 2.71**

**`Math.log(2.71)` returns 1.0**

**`Math.pow(2,3)` returns 8.0**

**`Math.pow(3,2)` returns 9.0**

**`Math.pow(3.5,2.5)` returns  
22.91765**

**`Math.sqrt(4)` returns 2.0**

**`Math.sqrt(10.5)` returns 3.24**

# Rounding Methods

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

# Rounding Methods Examples

```
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
Math rint(2.0) returns 2.0
Math rint(-2.0) returns -2.0
Math rint(-2.1) returns -2.0
Math rint(2.5) returns 2.0 //returns even value as double
Math rint(-2.5) returns -2.0
Math.round(2.6f) returns 3 //round returns integers
Math.round(2.0) returns 2
Math.round(-2.0f) returns -2
Math.round(-2.6) returns -3
```

# Min(), max(), and abs()

- **max(a,b)** and **min(a,b)**  
Returns the maximum or minimum of two parameters.
- **abs(a)**  
Returns the absolute value of the parameter.

## Examples:

**Math.max(2,3)** returns 3

**Math.max(2.5,3)** returns 3.0

**Math.min(2.5,3.6)** returns 2.5

**Math.abs(-2)** returns 2

**Math.abs(-2.1)** returns 2.1

# Method `random()`

Generates a random double value greater than or equal to 0.0 and less than 1.0 (`0.0 <= Math.random() < 1.0`)

Examples:

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

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

In general,

`a + Math.random() * b`  $\longrightarrow$  Returns a random (real) number between a and a + b, excluding a + b.

`(int) (a + Math.random() * b)`  $\longrightarrow$  Returns a random (integer) number between a and a + b, excluding a + b.

# Generating Random Characters

Each character has a unique [Unicode](#) between 0 and FFFF in hexadecimal (65535 in decimal).

To generate a random character is to generate a random integer between 0 and 65535 using the following expression:

```
(int) (Math.random() * (65535 + 1))
```

Note:

Since `0.0 <= Math.random() < 1.0`, you have to add 1 to 65535



# Generating Random Characters

Lowercase letter: The Unicode for lowercase letters are consecutive integers starting from the Unicode for 'a', 'b', 'c', ..., and 'z'.

The Unicode for 'a' is `(int) 'a'`

A random integer between `(int)'a'` and `(int)'z'` is

```
(int) ((int) 'a' + Math.random() * ((int) 'z' - (int) 'a' + 1))
```

So, a random lowercase letter is:

```
(char) ('a' + Math.random() * ('z' - 'a' + 1))
```

To generalize, a random character between any two characters `ch1` and `ch2` with `ch1 < ch2` can be generated as follows:

```
(char) (ch1 + Math.random() * (ch2 - ch1 + 1))
```

**See recommended textbook, Appendix B, page 1266, for character set order.**

# Class RandomCharacter

```
// RandomCharacter.java: Generate random characters
public class RandomCharacter {
    /** Generate a random character between ch1 and ch2 */
    public static char getRandomCharacter(char ch1, char ch2) {
        return (char)(ch1 + Math.random() * (ch2 - ch1 + 1)); }

    /** Generate a random lowercase letter */
    public static char getRandomLowerCaseLetter() {
        return getRandomCharacter('a', 'z'); }

    /** Generate a random uppercase letter */
    public static char getRandomUpperCaseLetter() {
        return getRandomCharacter('A', 'Z'); }

    /** Generate a random digit character */
    public static char getRandomDigitCharacter() {
        return getRandomCharacter('0', '9'); }

    /** Generate a random character */
    public static char getRandomCharacter() {
        return getRandomCharacter('\u0000', '\uFFFF'); }
}
```

# Class RandomCharacter

```
// Test class RandomCharacters
// class RandomCharacters methods are all static
public class TestRandomCharacters
{
    public static void main(String[] args)
    {
        System.out.print("A random character between 'a' and 'z' is: ");
        System.out.println(RandomCharacter.getRandomLowerCaseLetter());

        System.out.print("A random character between 'A' and 'Z' is: ");
        System.out.println(RandomCharacter.getRandomUpperCaseLetter());

        System.out.print("A random character between '0' and '9' is: ");
        System.out.println(RandomCharacter.getRandomDigitCharacter());

        System.out.print("A random character between 'g' and 'm' is: ");
        System.out.println(RandomCharacter.getRandomCharacter('g', 'm'));

        System.out.print("A random character between '3' and '7' is: ");
        System.out.println(RandomCharacter.getRandomCharacter('3', '7'));

        System.out.print("A random character between '!' and '*' is: ");
        System.out.println(RandomCharacter.getRandomCharacter('!', '*'));
    }
}
```

# 3. Character Data Type

A `char` variable stores a single character.

Character literals are delimited by single quotes:

`'a'`    `'X'`    `'7'`    `'$'`    `','`    `'\n'`    `'\t'`

Example declarations:

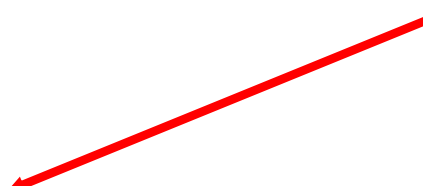
```
char topGrade = 'A';
```

```
char terminator = ';', separator = ' ';
```

Note the distinction between a primitive `char` variable, which holds only one character, and a `String` object, which can hold multiple characters.

# Character Type - Revisited

```
char letter = 'A';  
char numChar = '4';  
char letter = '\u0041'; //Unicode for A  
char numChar = '\u0034'; //Unicode for character 4
```



Four hexadecimal digits.

**NOTE:** The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character. For example, the following statements display character b.

```
char ch = 'c'; //c ASCII code is 99  
ch = ch + 1;  
System.out.println(ch); //prints character d  
ch = ch - 2;  
System.out.println(ch); //prints character b
```

# ASCII Code in Decimal

TABLE B.1 ASCII Character Set in the Decimal Index

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	\$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~	del		

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

# Casting char Type

```
int i = 'a'; //Same as int i = (int)'a'; which is 97
```

```
char ch = 97; //Same as char ch = (char)97; which is 'a'
```

# Comparing char Type

```
if (ch >= 'A' && ch <= 'Z')  
    System.out.println(ch + " is an uppercase letter");  
else if (ch >= 'a' && ch <= 'z')  
    System.out.println(ch + " is a lowercase letter");  
else if (ch >= '0' && ch <= '9')  
    System.out.println(ch + " is a numeric character");
```



# Class Character Methods

Method	Description
<code>isDigit(ch)</code>	Returns true if the specified character is a digit.
<code>isLetter(ch)</code>	Returns true if the specified character is a letter.
<code>isLetterOrDigit(ch)</code>	Returns true if the specified character is a letter or digit.
<code>isLowerCase(ch)</code>	Returns true if the specified character is a lowercase letter.
<code>isUpperCase(ch)</code>	Returns true if the specified character is an uppercase letter.
<code>toLowerCase(ch)</code>	Returns the lowercase of the specified character.
<code>toUpperCase(ch)</code>	Returns the uppercase of the specified character.

# Class Character Methods

Class Character in the **Language package**, automatically imported.

```
Character ch1 = new Character('b'); //object NOT char type  
Character ch2 = new Character('9'); //object NOT char type
```

<code>Character.isLowerCase(ch1)</code>	returns true
<code>Character.isLetterOrDigit(ch1)</code>	returns true
<code>Character.isDigit(ch1)</code>	returns false
<code>Character.isDigit(ch2)</code>	returns true
<code>Character.toUpperCase(ch1)</code>	returns B

# Class Character Test

```
// Class Character Test
import java.util.Scanner;
public class CharacterTest
{
    public static void main (String[] args)
    {
        Character ch1 = new Character('b'); //object NOT char type
        Character ch2 = new Character('9'); //object NOT char type

        System.out.println(Character.isLowerCase(ch1)); //returns true
        System.out.println(Character.isLetterOrDigit(ch1)); //returns true
        System.out.println(Character.isDigit(ch1)); //returns false
        System.out.println(Character.isDigit(ch2)); //returns true
        System.out.println(Character.toUpperCase(ch1)); //returns B

        char ch3 = 'R'; // char type variable
        char ch4 = '7'; // char type variable
        char ch5 = '*'; // char type variable

        System.out.println(Character.isLowerCase(ch3)); //returns false
        System.out.println(Character.isLetterOrDigit(ch5)); //returns false
        System.out.println(Character.isDigit(ch4)); //returns true
        System.out.println(Character.isDigit(ch5)); //returns false
        System.out.println(Character.toLowerCase(ch3)); //returns r
    }
}
```

# Escape Sequences

<i>Description</i>	<i>Escape Sequence</i>	<i>Unicode</i>
Backspace	\b	\u0008
Tab	\t	\u0009
Linefeed	\n	\u000A
Carriage return	\r	\u000D
<hr/>		
Backslash	\\	\u005C
Single Quote	\'	\u0027
Double Quote	\"	\u0022

## 4. Class *String*

- To create a *String* object, we need to declare a variables of type *String*:

```
String title = "Java Software Solutions";
```

- Each string literal (enclosed in double quotes) represents a *String* object
- Once a *String* object has been created, **neither its value nor its length can be changed**. Thus, *String* objects are *immutable*
- The *String* type is **not a primitive type**. It is a class type and known as a *object* or *reference type*.

# String Methods

- However, several methods of the `String` class return new `String` objects that are modified versions of the original string
- A `String` object is a sequence of characters (known as Single-Dimensional Array).

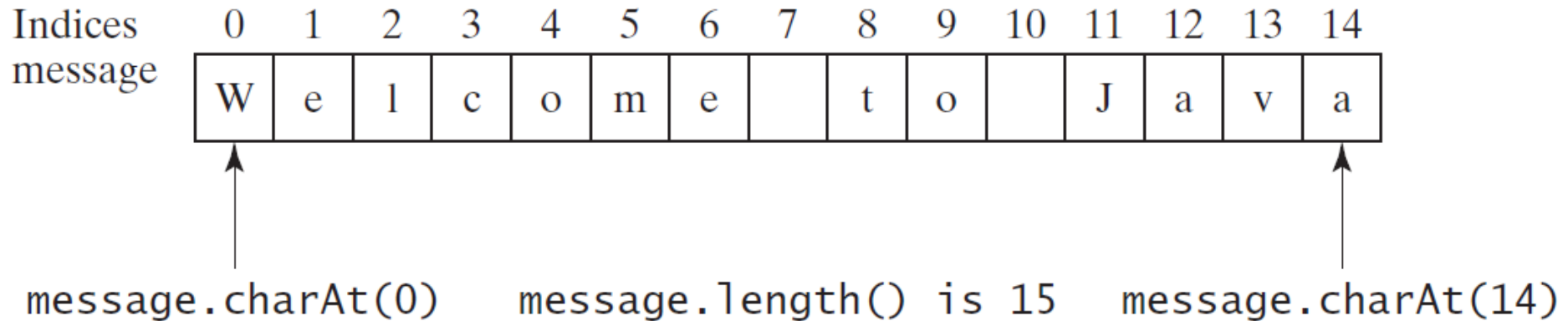
```
String courseName = "CS 1301";
```

0	1	2	3	4	5	6
C	S		1	3	0	1

# String Index Values

- It is occasionally helpful to refer to a particular character within a string
- This can be done by specifying the character's numeric *index* (position)
- The indexes begin at **zero** in each string
- In the string "Hello", the character 'H' is at index **0** and the 'o' is at index **4**

# Getting Characters from a String



```
String message = "Welcome to Java";
```

```
char ch = message.charAt(0);
```

```
System.out.println("The first character in message is " + ch);
```

```
String message = "Welcome to Java";
```

```
int messageLength = message.length();
```

```
System.out.println("The length of message is " + messageLength);
```



# String Concatenation

// Three strings are concatenated, notice the spaces

```
String message = "Welcome " + "to " + "Java";
```

// String Chapter is concatenated with number 2

```
String name1 = "Chapter" + 2; //name1 holds Chapter2
```

// String Supplement is concatenated with character B

```
String name2 = "Supplement" + 'B'; //name2 holds SupplementB
```

# Example

```
public class StringMutation
{
    // Prints a string and various mutations of it.
    public static void main (String[] args)
    {
        String phrase = "Change is inevitable";
        String mutation1, mutation2, mutation3, mutation4;

        System.out.println ("Original string: \"\" + phrase + "\"");
        System.out.println ("Length of string: \" + phrase.length());

        mutation1 = phrase.concat(", except from vending machines.");
        mutation2 = mutation1.toUpperCase();
        mutation3 = mutation2.replace ('E', 'X');
        mutation4 = mutation3.substring (3, 30); //excluding position 30
        System.out.println ("Mutation #1: \" + mutation1);
        System.out.println ("Mutation #2: \" + mutation2);
        System.out.println ("Mutation #3: \" + mutation3);
        System.out.println ("Mutation #4: \" + mutation4);

        System.out.println ("Mutated length: \" + mutation4.length());
    }
}
```

# Example

## Output:

Original string: "Change is inevitable"

Length of string: 20

Mutation #1: Change is inevitable, except from vending machines.

Mutation #2: CHANGE IS INEVITABLE, EXCEPT FROM VENDING MACHINES.

Mutation #3: CHANGX IS INXVITABLX, XXCXPT FROM VXNDING MACHINXS.

Mutation #4: NGX IS INXVITABLX, XXCXPT F

Mutated length: 27

# Other Common String Methods

```
String S1 = "Welcome";
String S2 = new String(char[]);
S2 = "    Hello!    ";
char ch = S1.charAt(index);
int length = S1.length();
int index = S1.indexOf(ch);
int index = S1.lastIndexOf(ch);
boolean b = S1.equals(S2);
boolean b = S1.equalsIgnoreCase(S2);
boolean b = S1.startsWith(S2);
Boolean b = S1.endsWith(S2);
String S = S1.toUpperCase();
String S = S2.toLowerCase();
String S = S2.substring(i); //from position i to last position
String S = S2.substring(i,j); //excluding j position
String S = S2.replace(ch1,ch2);
String S = S2.trim(); //returns "Hello!", no spaces
```

# Reading Strings

```
Scanner input = new Scanner(System.in);  
System.out.print("Enter three words separated by spaces: ");  
String s1 = input.next();  
String s2 = input.next();  
String s3 = input.next();  
System.out.println("First word is " + s1);  
System.out.println("Second word is " + s2);  
System.out.println("Third word is " + s3);
```

**Note:** If we use

```
String s1 = input.nextLine();
```

`s1` contains all typed characters until we press the "Enter" key.

# Reading Characters

//Characters are read as strings

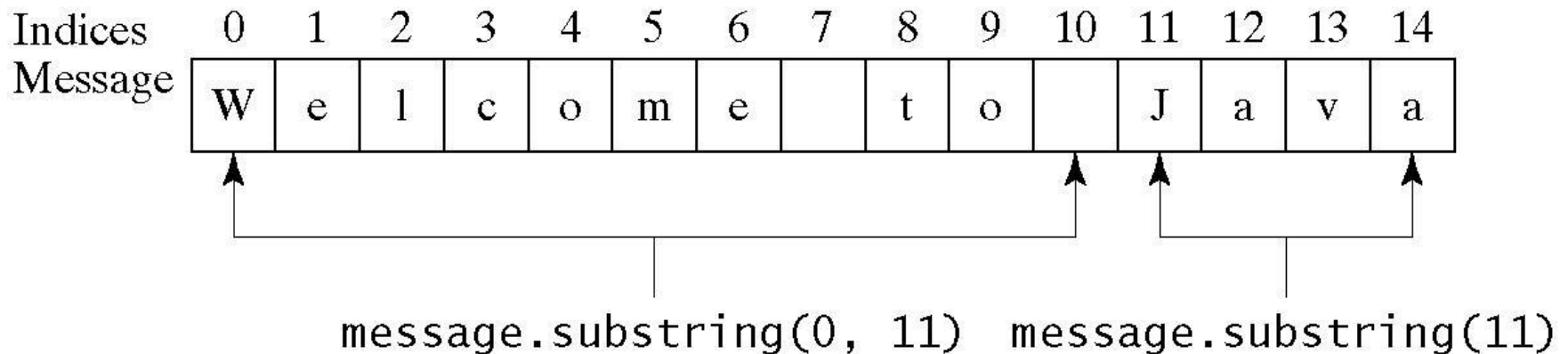
```
Scanner input = new Scanner(System.in);  
System.out.print("Enter a character: ");  
String s = input.nextLine(); //must press the Enter key  
char ch = s.charAt(0);  
System.out.println("The entered character is " + ch);
```

# Comparing Strings

Method	Description
<code>compareTo(s1)</code>	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 <code>s1</code> .
<code>compareToIgnoreCase(s1)</code>	<p>Same as <code>compareTo</code> except that the comparison is case insensitive.</p> <p>Returns true if this string starts with the specified prefix.</p> <p>Returns true if this string ends with the specified suffix.</p>

# Obtaining Substrings

Method	Description
<code>substring(beginIndex)</code>	Returns this string's substring that begins with the character at the specified <code>beginIndex</code> and extends to the end of the string, as shown in Figure 4.2.
<code>substring(beginIndex, endIndex)</code>	Returns this string's substring that begins at the specified <code>beginIndex</code> and extends to the character at index <code>endIndex - 1</code> , as shown in Figure 9.6. Note that the character at <code>endIndex</code> is not part of the substring.





# indexOf () method

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

# Conversion of Strings/Numbers

You can convert strings of digits to numbers:

```
String intString = "123";  
int intValue = Integer.parseInt(intString);
```

```
String doubleString = "123.456";  
double doubleValue = Double.parseDouble(doubleString);
```

You can convert numbers to strings:

```
int number = 123456;  
String s = "" + number; //gives "123456"
```

# 5. `printf()` Statement

Use the `printf` statement.

```
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.

# Frequently-Used Specifiers

Specifier	Output	Example
<code>%b</code>	a boolean value	true or false
<code>%c</code>	a character	'a'
<code>%d</code>	a decimal integer	200
<code>%f</code>	a floating-point number	45.4600000
<code>%e</code>	a standard scientific notation	4.556000e+01
<code>%s</code>	a string	"Java is cool"

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

The diagram illustrates the argument list for the `printf` method. A bracket labeled "items" spans the arguments `count` and `amount`. An arrow points from `count` to the `%d` specifier, and another arrow points from `amount` to the `%f` specifier.

Output: count is 5 and amount is 45.5600000

# Examples

## Examples:

- `%7b` prints true or false with 2 or 3 leading spaces
- `%3c` prints character with 2 leading spaces
- `%6d` prints integer with leading spaces if less than 6 digits. Else, the entire number is printed.
- `%3.2f` prints real number with 2 digits after the decimal point.
- `%3.2e` prints real number with 2 digits after the decimal point and the exponent part.
- `%10s` prints a string in 10 space, with leading spaces if less than 10 characters.

# Examples

## Examples:

```
System.out.printf("%6d%6s%6.1f\n", 123, "ABC", 12.37);  
System.out.printf("%-6d%-6s%-6.1f\n", 123, "ABC", 12.37);
```

Output: (each - represents one space)

```
---123---ABC--12.4
```

```
123---ABC---12.4--
```

**Recommended Homework:** Type and run program [FormatDemo](#), from the recommended textbook, listing 4.6, page 148. It shows how to display tabulated outputs using **printf()** statement.

**End of Chapter 4**