Chapter 8

Characters and Strings

OBJECTIVES
After you have read and studied this chapter, you should be able to

- Declare and manipulate data of the char data type.
- Write string processing programs using String and StringBuffer objects.
- Differentiate the String and StringBuffer classes and use the correct class in solving a given task.
- Distinguish the primitive and reference data types and show how the memory allocation between the two is different.
- Tell the difference between equality and equivalence testings for String objects.
- Show, by using the state-of-memory diagrams, how objects are passed to methods and returned from methods.

TABLE 8.1 ASCII Codes.

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
10	lf	vt	ff	cr	so	si	dle	dc1	dc2	dc3
20	cd4	nak	syn	etb	can	em	sub	esc	fs	gs
30	rs	us	sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	}	|	~	del		

FIGURE 8.1 An indexed expression is used to refer to individual characters in a string.

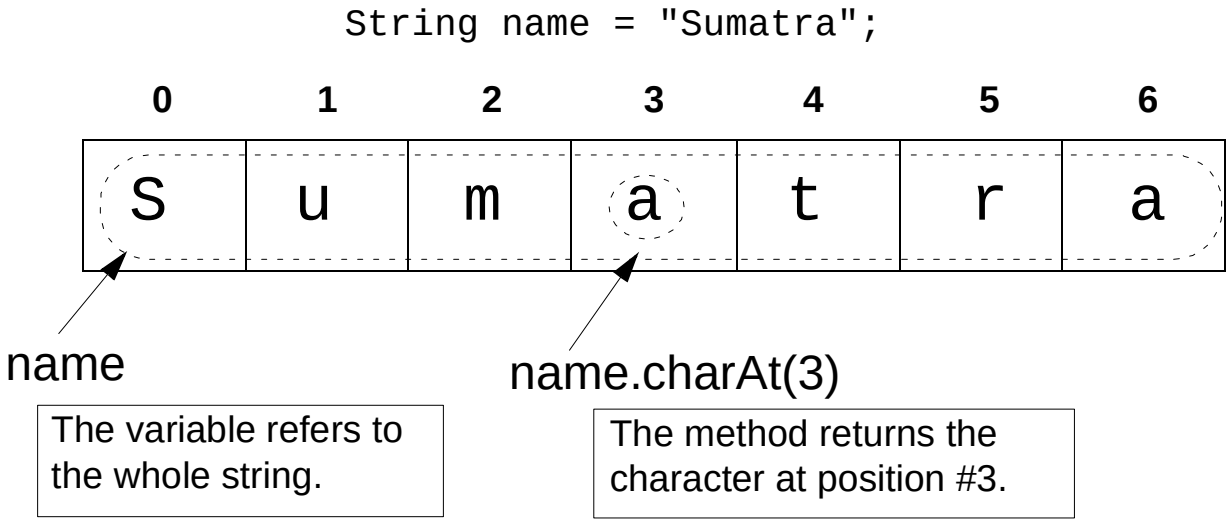


FIGURE 8.2 The taxonomy of data types.

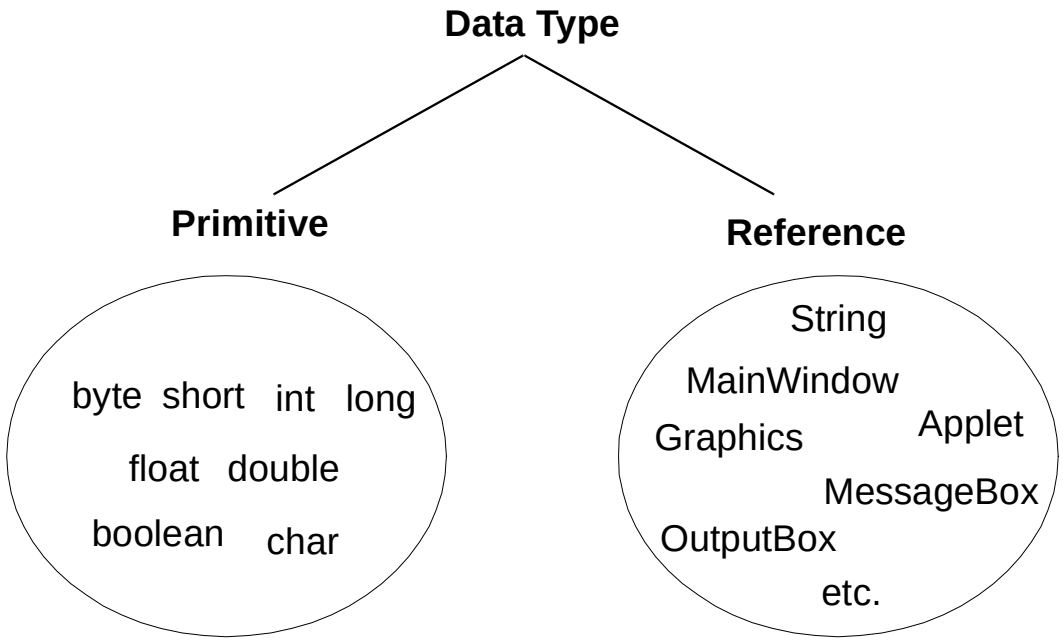


FIGURE 8.3 Effect of assigning an integer value to a variable.

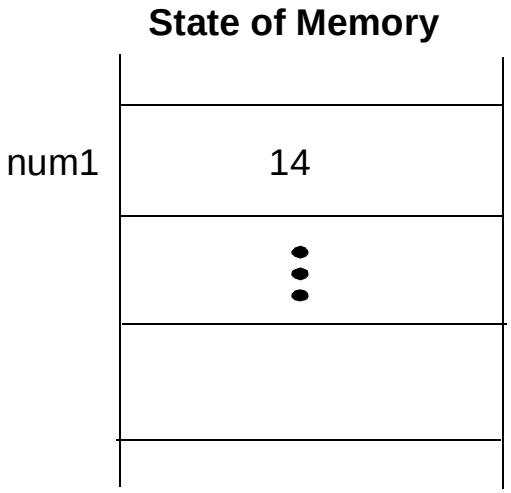


FIGURE 8.4 Effect of assigning values to integer variables.

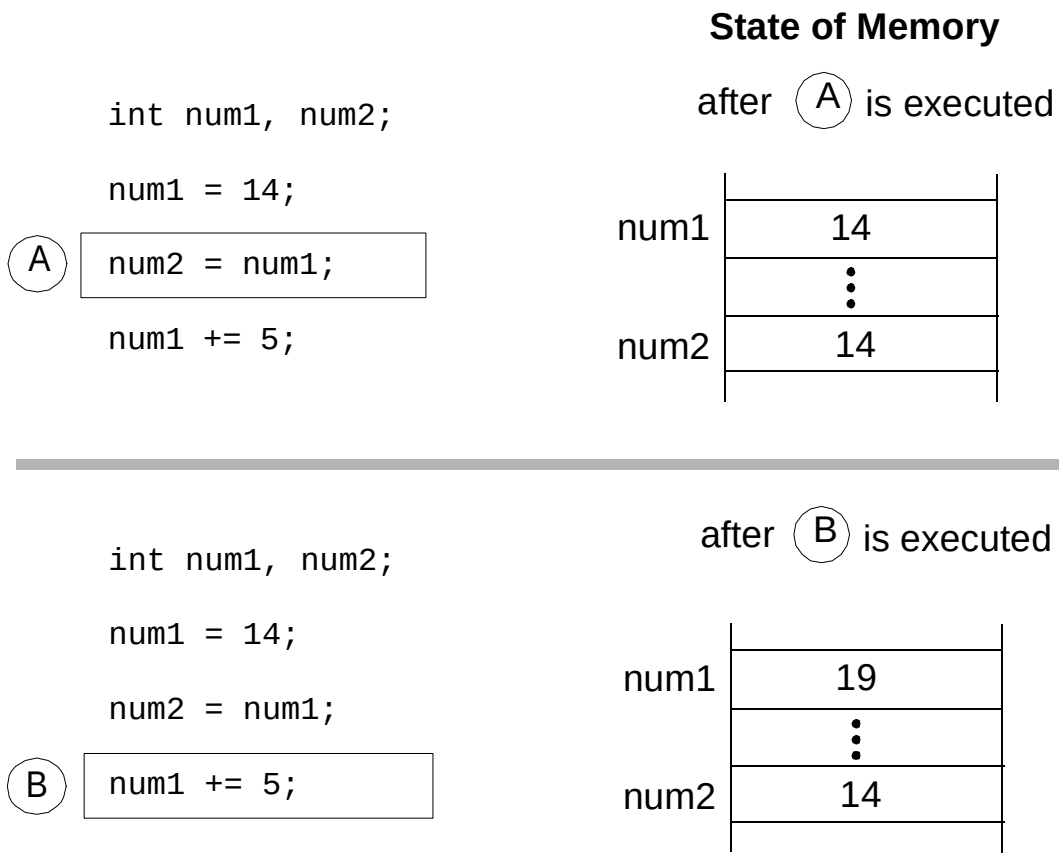


FIGURE 8.5 Effect of assigning a **String** value to a variable.

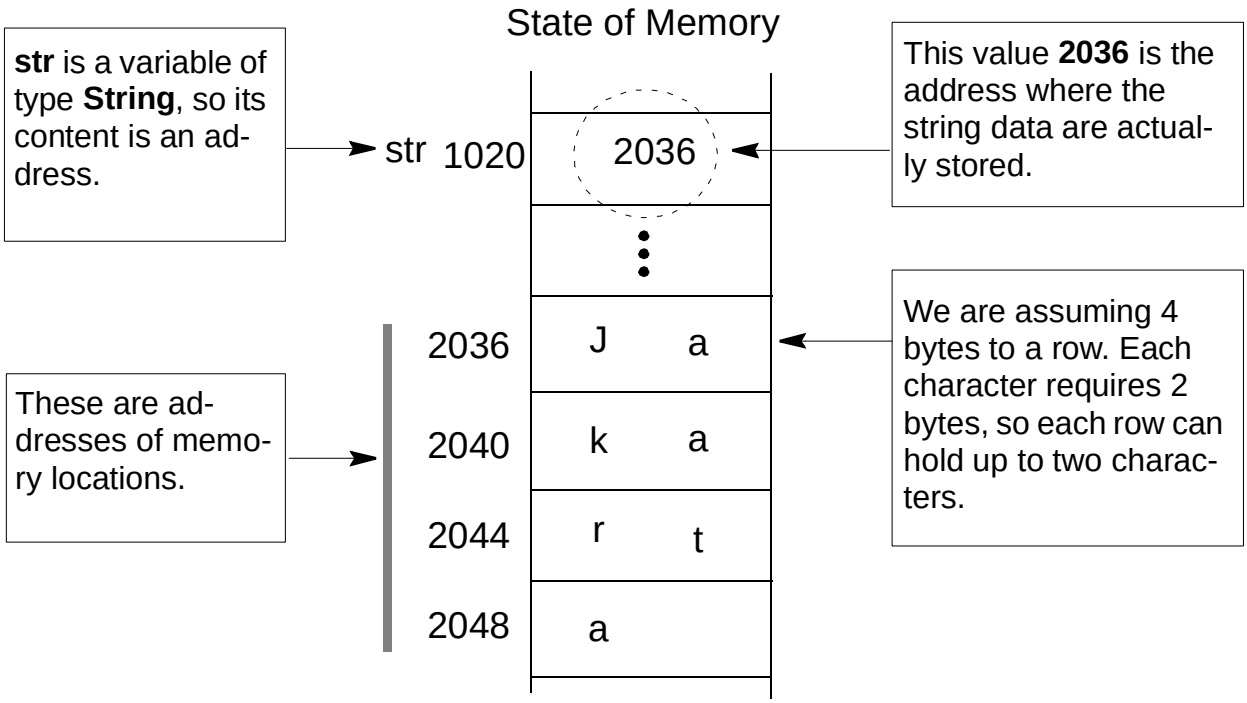


FIGURE 8.6 Hypothetical memory management scheme for storing a **String** value.

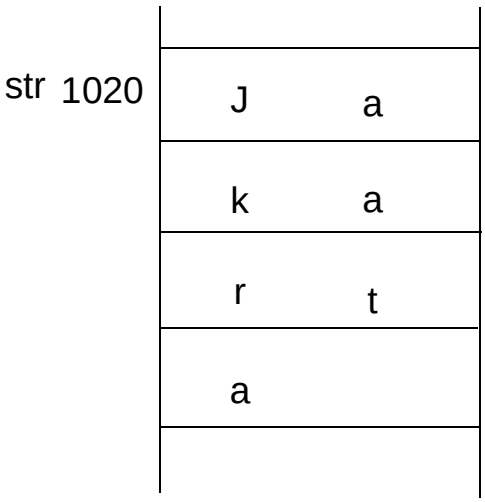


FIGURE 8.7 Memory shortage problem with the hypothetical memory management scheme.

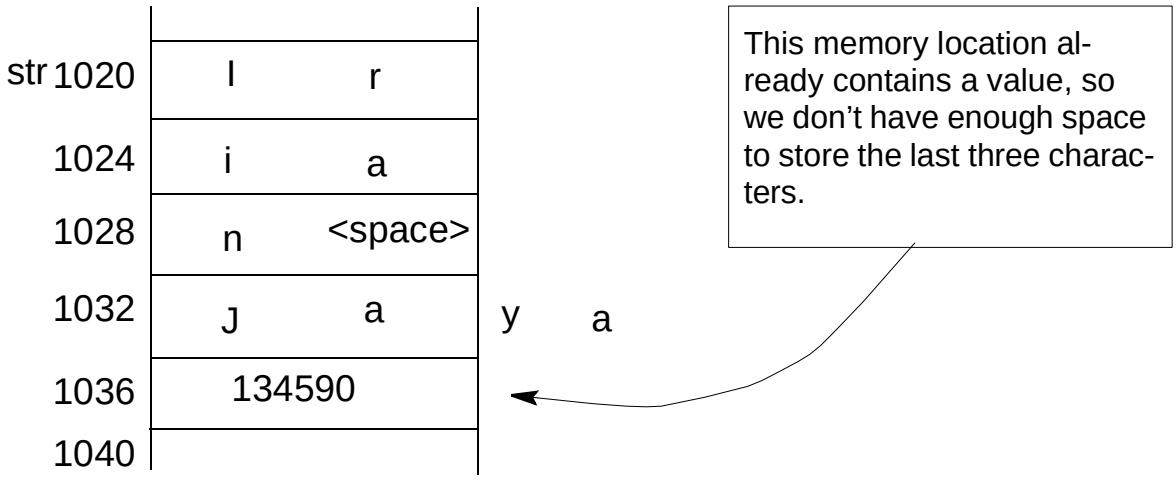
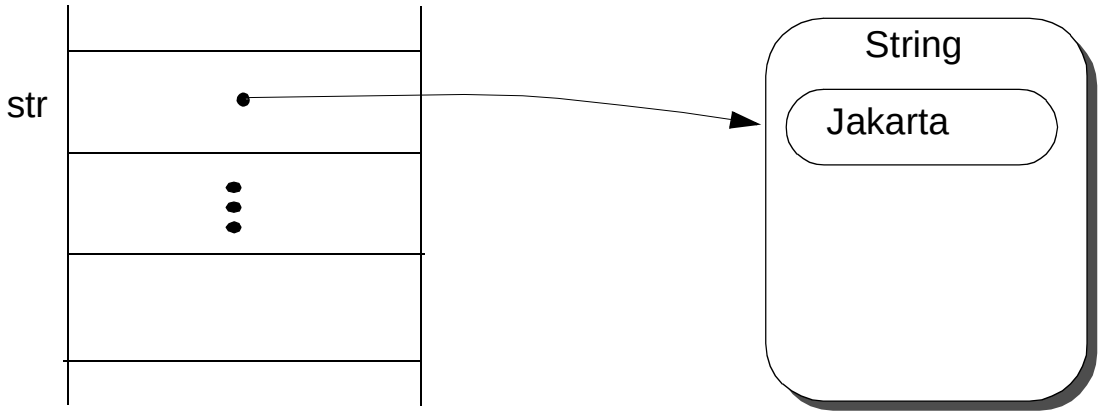
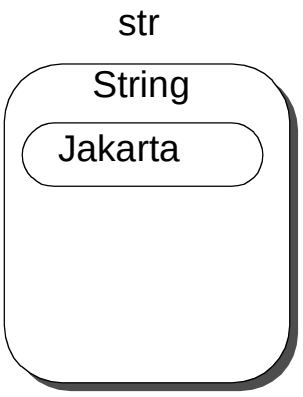


FIGURE 8.8 A preferred style of diagram for representing memory allocation for a reference data type.





Object Diagram Representation

FIGURE 8.9 Effect of assignment statements on reference data type **String**.

Both **word1** and **word2** are allocated memory, but no actual objects are created yet, so both are **null**.

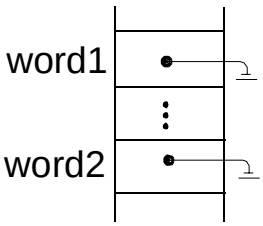
State of Memory

after (A) is executed

(A)
```
String word1, word2;

word1 = new String("Java");

word2 = word1;
```



Both **word1** and **word2** are allocated memory, but no actual objects are created yet, so both are **null**.

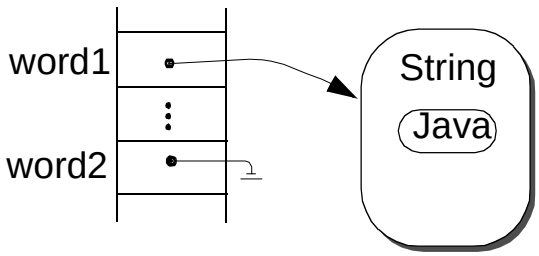
One **String** object is created and assigned to **word1**; that is, **word1** points to the object.

after (B) is executed

```
String word1, word2;

(B) word1 = new String("Java");

word2 = word1;
```



Content of **word1** (which is a reference to the **String** object) is assigned to **word2**, making **word2** also point to the same **String** object.

after (C) is executed

```
String word1, word2;

word1 = new String("Java");

(C) word2 = word1;
```

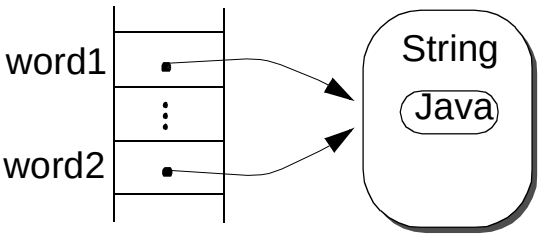
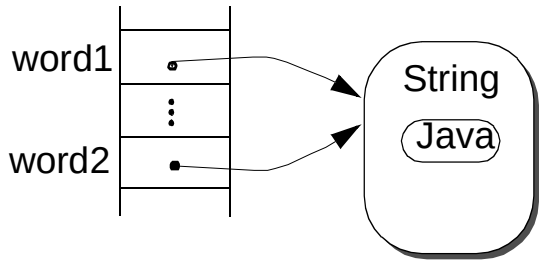


FIGURE 8.10 The difference between the equality test and the equals method.

Case A: Referring to the same object.

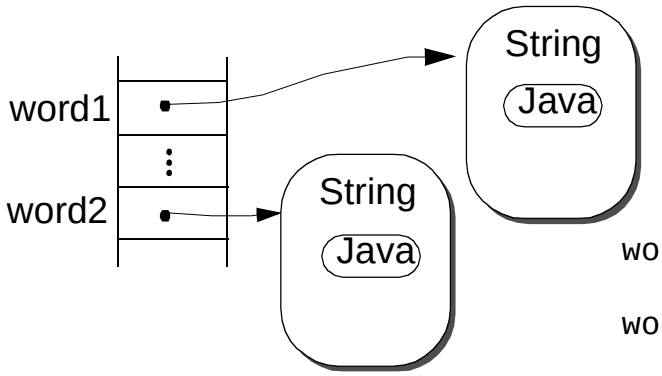



```
word1 == word2          is true

word1.equals( word2 )   is true
```

Note: If **x == y** is **true**, then **x.equals(y)** is also **true**. The reverse is not always true.

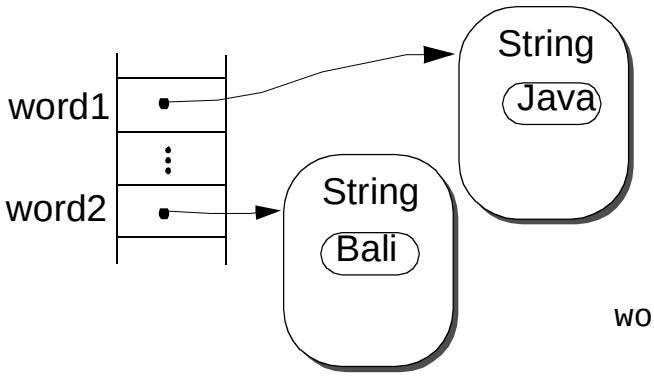
Case B: Referring to different objects having identical string values.




```
word1 == word2          is false

word1.equals( word2 )   is true
```

Case C: Referring to different objects having different string values.




```
word1 == word2          is false

word1.equals( word2 )   is false
```

FIGURE 8.11 How the memory space for parameters are allocated and deallocated.

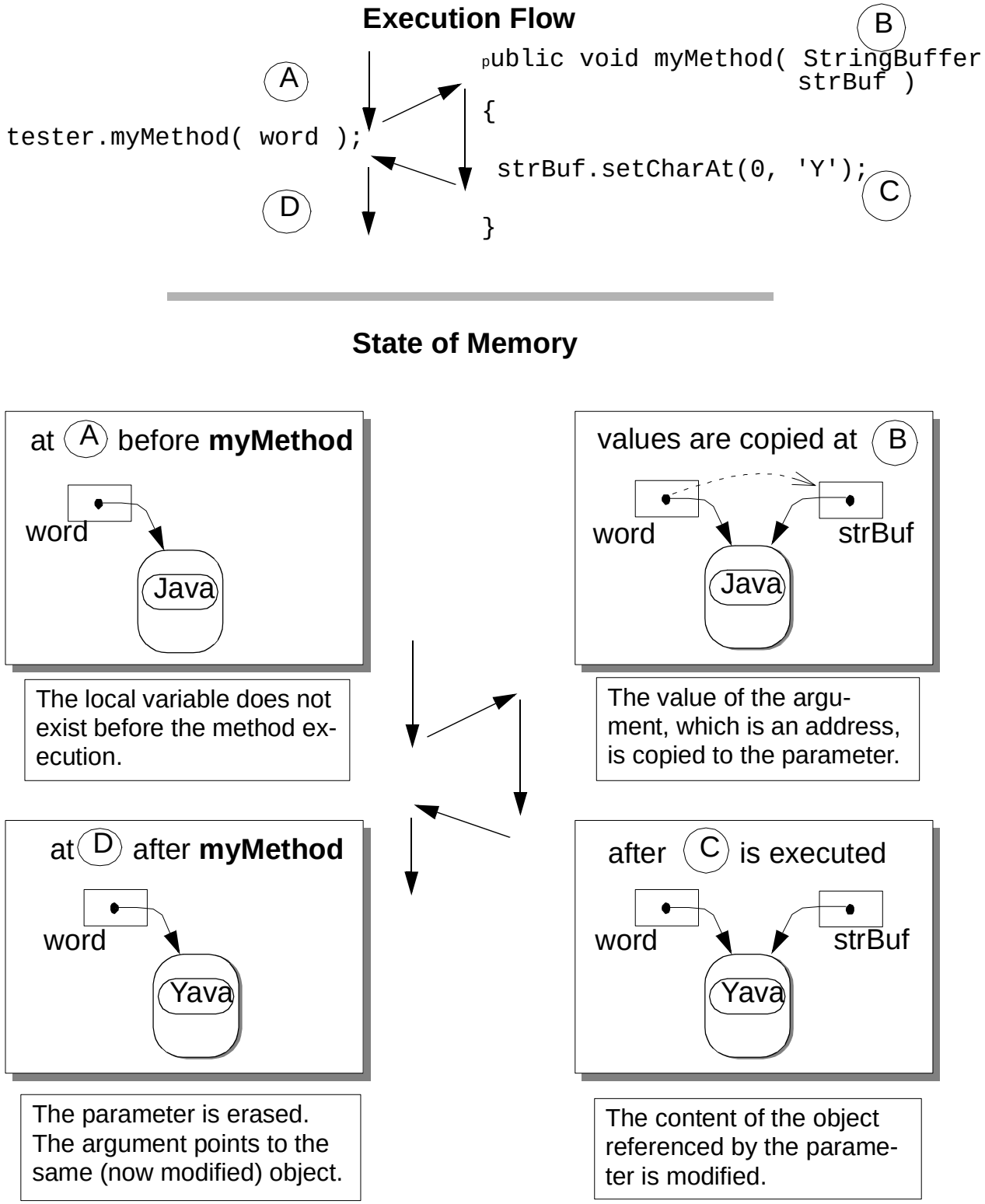


FIGURE 8.12 How a local copy of the passed object is created and manipulated. The original object will not change.

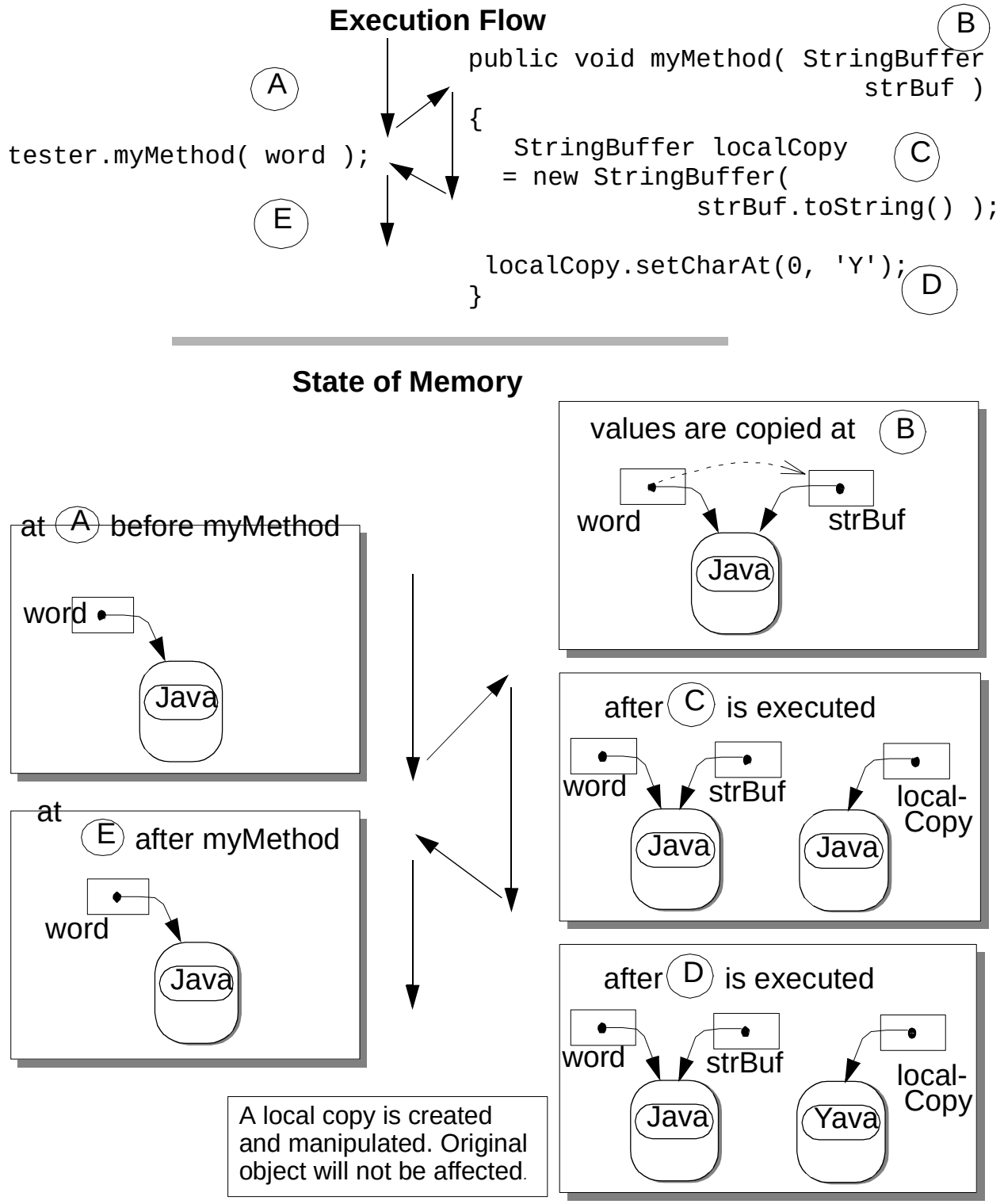


FIGURE 8.13 How the reference data type is returned from the **sayHI** method.

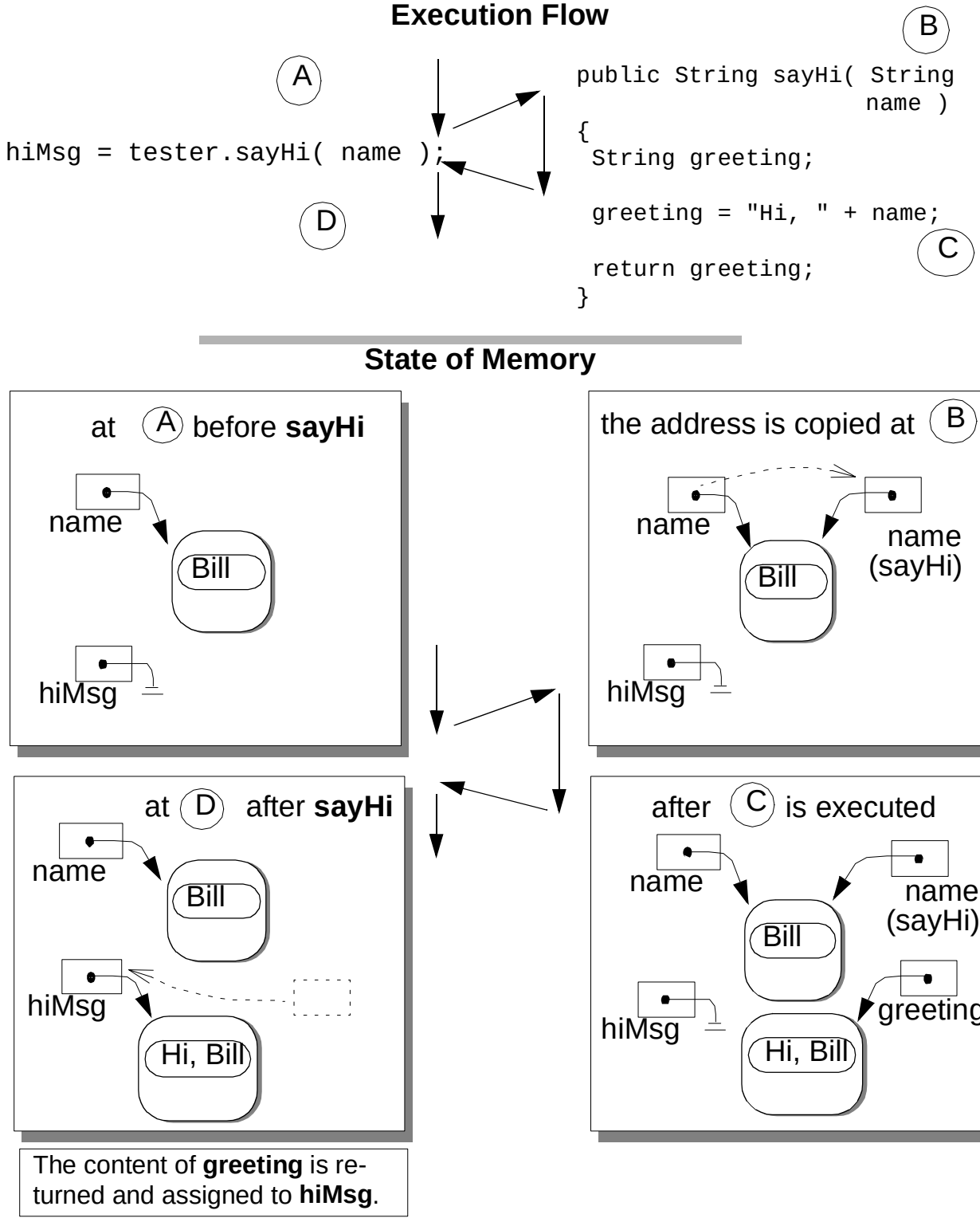


FIGURE 8.14 The object diagram for the **EggyPeggy** program. Note: **String** and **StringBuffer** objects are not shown here.