

Introduction to Computing *Using Java*

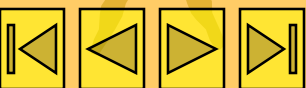
Tutorial 8

String



Data Structure

- ★ With the 8 primitive types, we can only store limited kinds of data.
- ★ With the class and object concept, we can *create* and *use* other more advanced data structures for our information needs.
- ★ There have been many commonly-used ones already defined, e.g. String





String

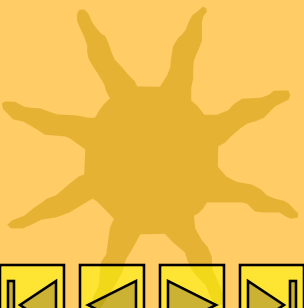


- ★ The *String class* defines a convenient data type for storing and manipulating a sequence of characters, i.e. text.



- ★ Remember how to create an object?

```
<classname>    <variable_or_field>;  
<variable_or_field> = new <classname> ( ) ;
```



- ★ Similarly, we can create a *String* object:

```
String myName;  
myName = new String();
```





"No New?"



★ *New-less* String object creation!!!

```
String myName;
```

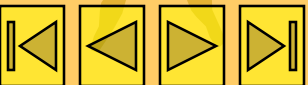
```
myName = "*** Michael Fung! Hi :)";
```



★ Yes, " " creates you a *new String object* with constant content.



★ This short-hand is dedicated to class String only!

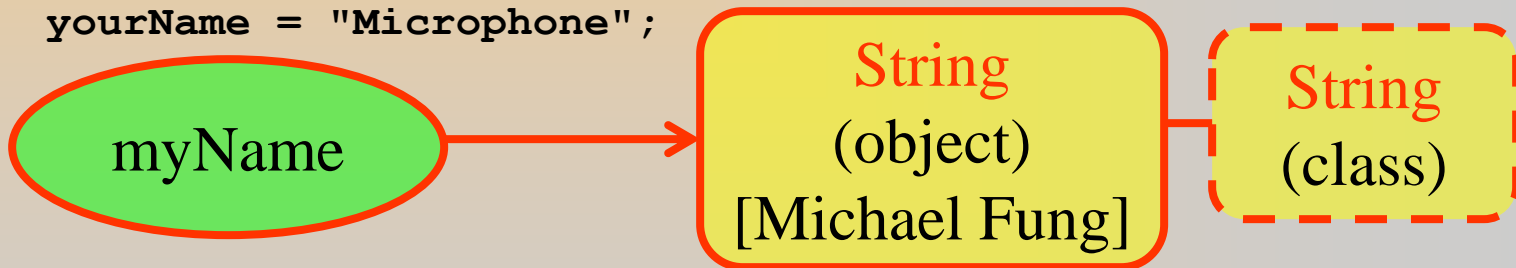




String Properties

★ Strings are objects, not primitives!

```
int i = 999;  
int j = i;           // copy value of i (999) to j  
String myName = "Michael Fung";  
String yourName = myName; // copy object reference  
yourName = "Microphone";
```

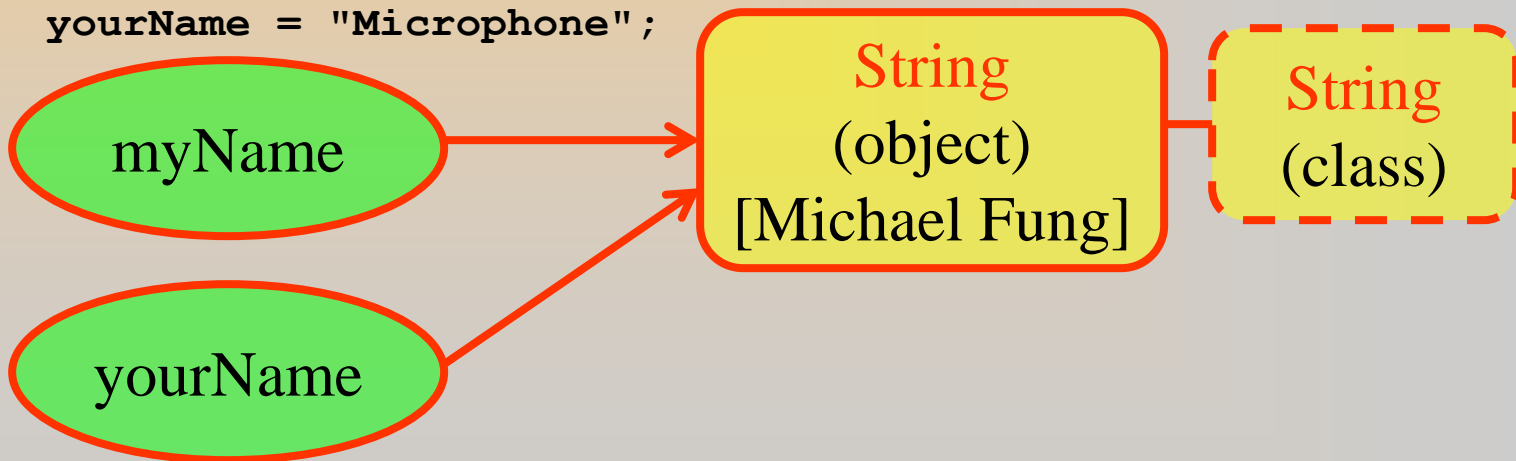




String Properties

★ Strings are objects, not primitives!

```
int i = 999;  
int j = i;           // copy value of i (999) to j  
String myName = "Michael Fung";  
String yourName = myName; // copy object reference  
yourName = "Microphone";
```

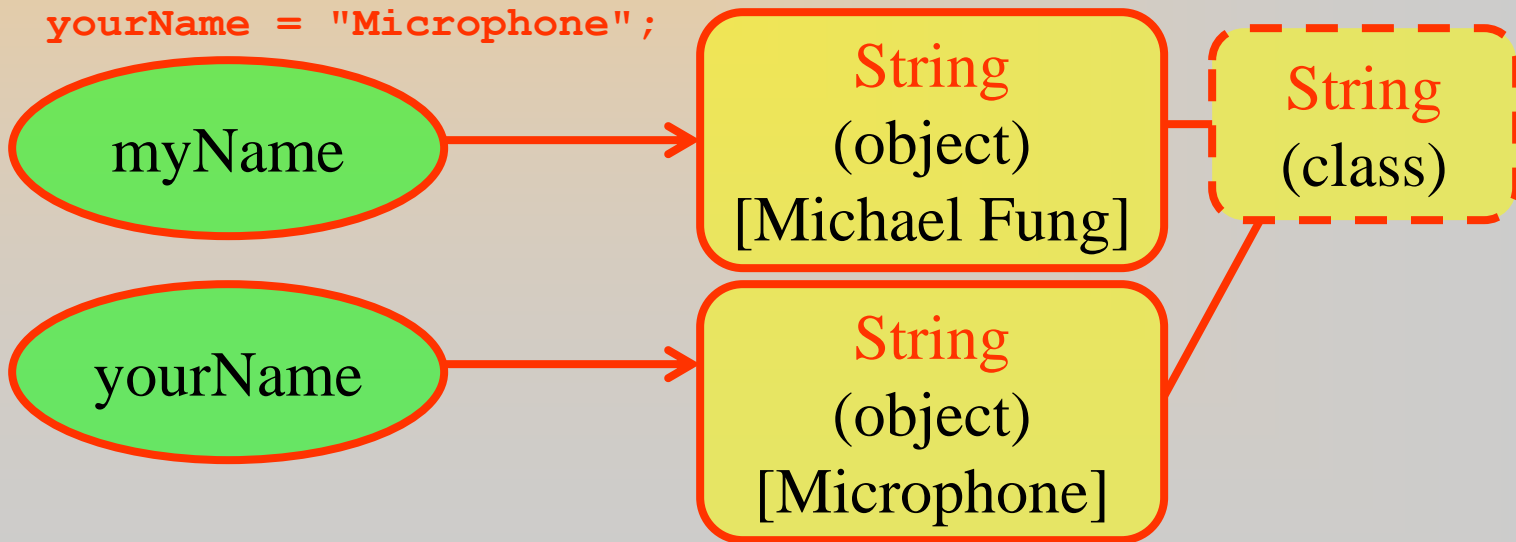




String Properties

★ Strings are objects, not primitives!

```
int i = 999;  
int j = i;           // copy value of i (999) to j  
String myName = "Michael Fung";  
String yourName = myName; // copy object reference  
yourName = "Microphone";
```





String Properties

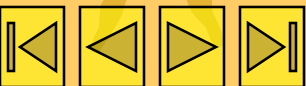
- ★ String objects can be concatenated (joined together) using a special operator ‘+’

```
String myName    = "Michael Fung";  
String yourName  = "Microphone";  
String concat    = myName + " is...";  
System.out.println(concat);  
String hello     = concat + yourName;  
System.out.println(hello);
```

Michael Fung is...

Michael Fung is...Microphone

- ★ Class String is privileged to have the ‘+’.





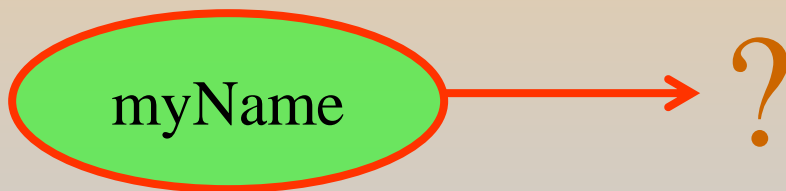
String Properties



- ★ Content of a **String** object could be *empty*.

```
String myName;  
myName = "";  
String yourName;
```

- ★ *null* object reference \neq **String** object ""



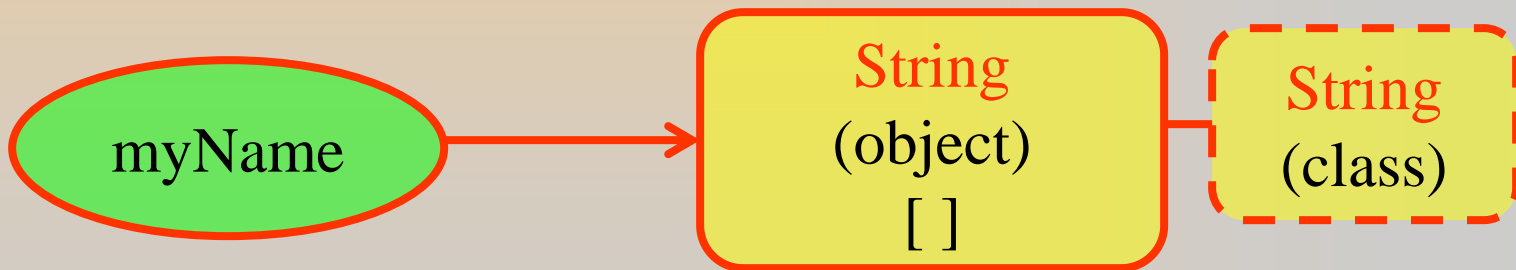


String Properties

- ★ Content of a **String** object could be *empty*.

```
String myName;  
myName = "";  
String yourName;
```

- ★ *null* object reference \neq **String** object ""





String Properties

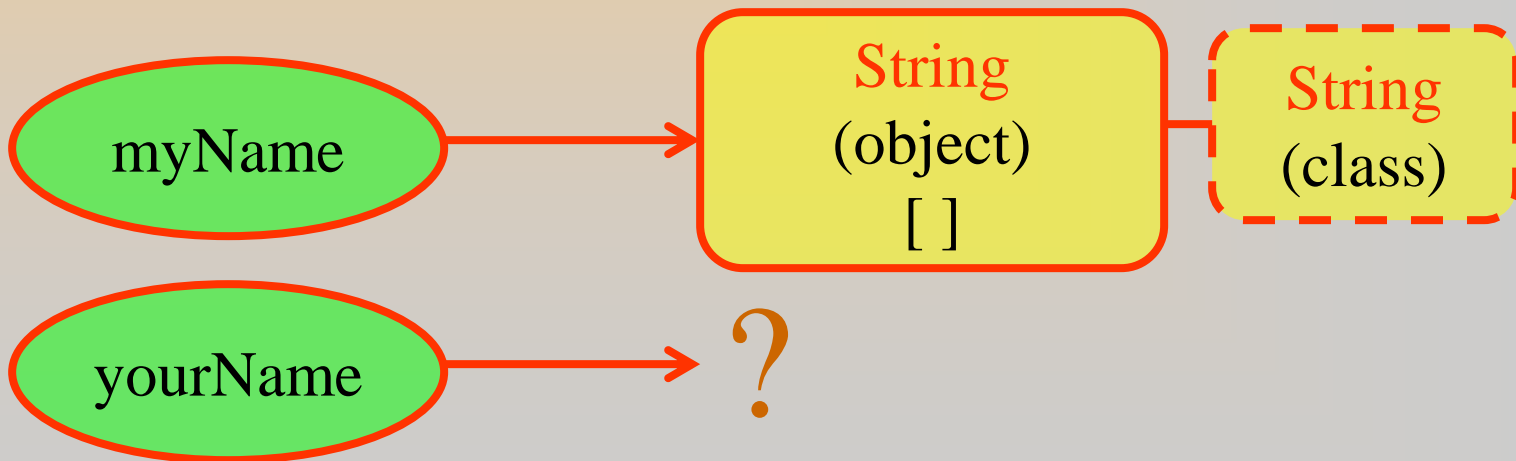
- ★ Content of a **String** object could be *empty*.

```
String myName;
```

```
myName = "";
```

```
String yourName; // no assignment means null
```

- ★ *null* object reference \neq **String** object ""

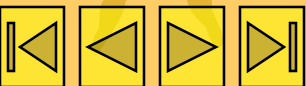




String Fields?!

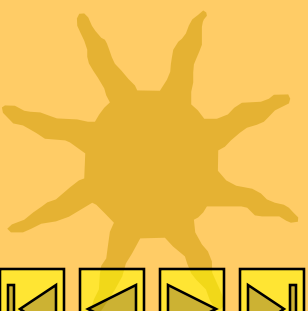


- ★ If we glance through the Java Reference Manual or Java books, we seldom find the names of the fields defined in class String!
- ★ They are **private** and inaccessible by us.
- ★ They are encapsulated and well-protected.





String Methods



★ To manipulate a String object, we must use the provided *methods*.

★ Under clever and considerate design, the method names are *self-explanatory*.

★ To get the *length* of a String object,

```
String myName = "Michael Fung";  
System.out.println(myName.length());           // message
```

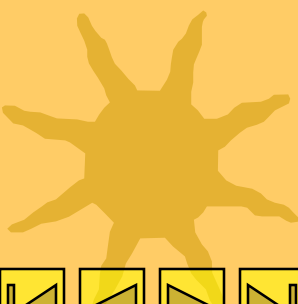
★ **Ooops...** to get the length of an array,

```
int[ ] even_numbers = {2, 4, 6, 8, 10, 12, 14, 16};  
System.out.println(even_numbers.length);       // property
```





String Methods



★ Print a String vertically!

```
String myName = "Michael";  
for (int i = 0; i < myName.length(); i++)  
    System.out.println(myName.charAt(i));
```

```
-----  
M  
i  
c  
h  
a  
e  
l
```

★ The instance method **char charAt(int)** returns you a **char** at the i^{th} position of the String object.



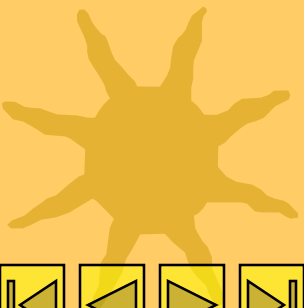
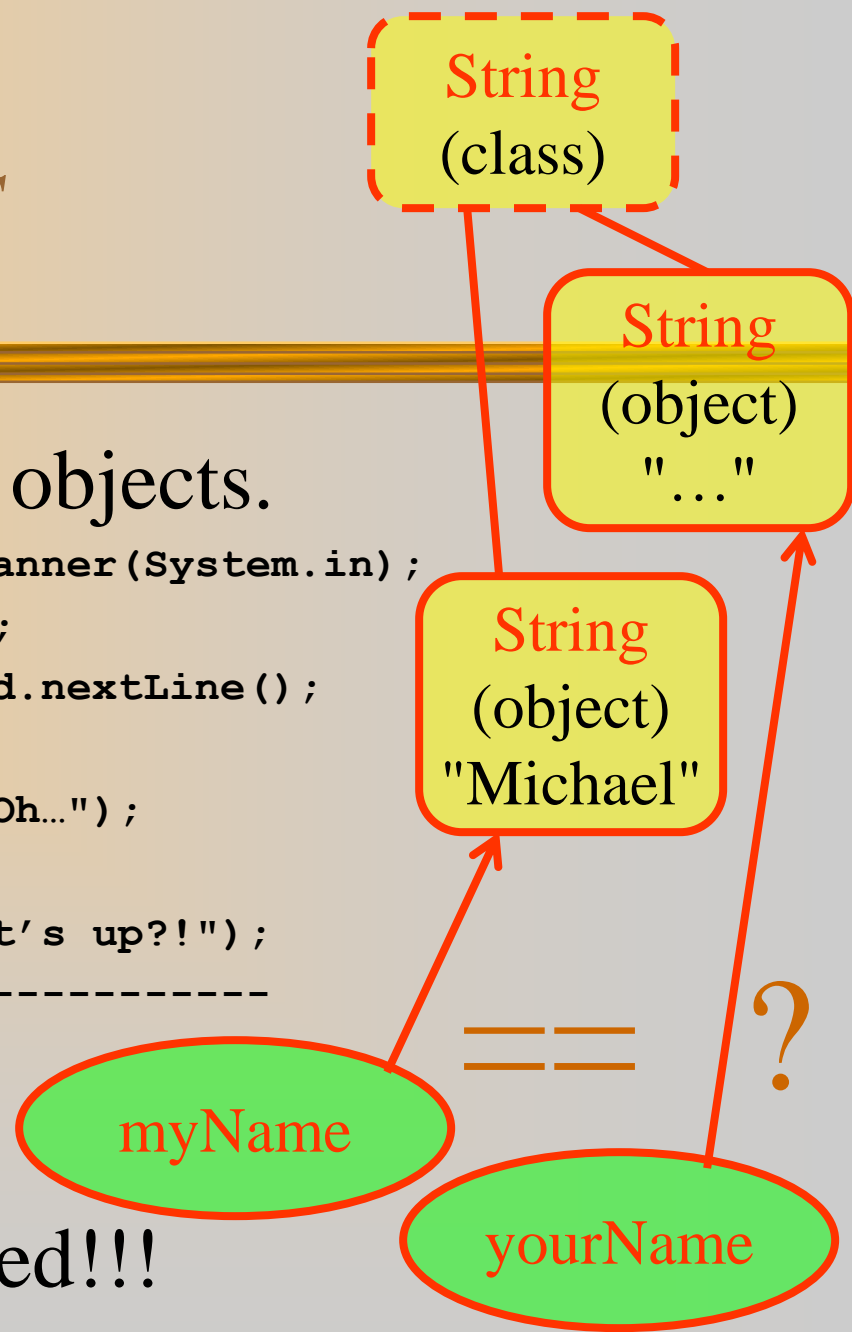


String Methods

★ Compares 2 String objects.

```
Scanner keyboard = new Scanner(System.in);  
String myName = "Michael";  
String yourName = keyboard.nextLine();  
if (myName == yourName)  
    System.out.println("Uk...Oh...");  
else  
    System.out.println("What's up?!");  
-----  
What's up?!
```

★ We are comparing the references indeed!!!



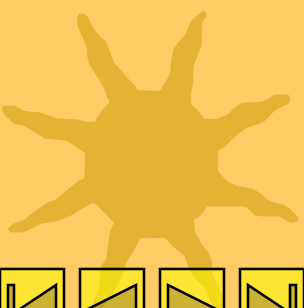
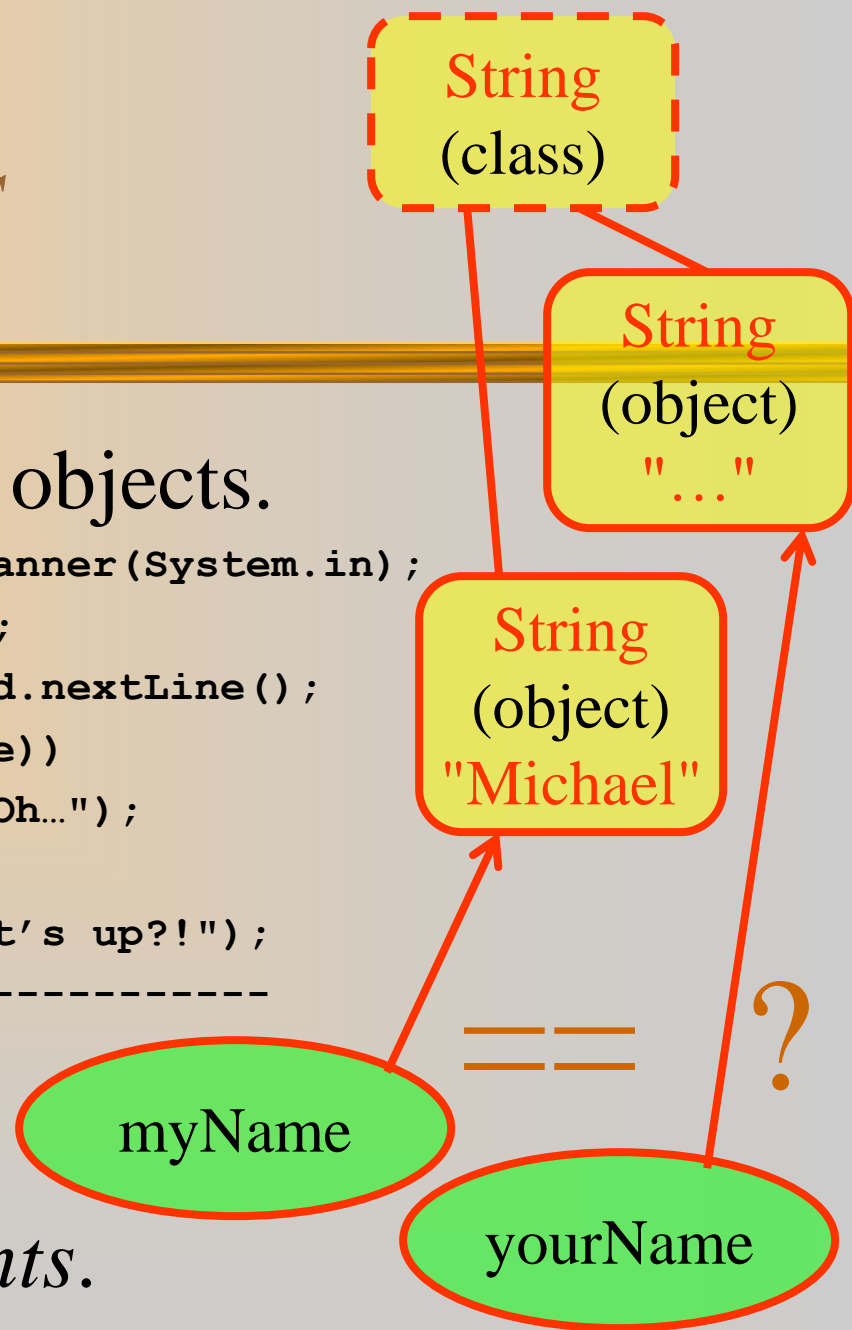


String Methods

★ Compares 2 String objects.

```
Scanner keyboard = new Scanner(System.in);  
String myName = "Michael";  
String yourName = keyboard.nextLine();  
if (myName.equals(yourName))  
    System.out.println("Uk...Oh...");  
else  
    System.out.println("What's up?!");  
-----  
Uk...Oh...
```

★ We want to compare the *contents*.

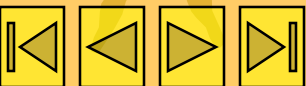
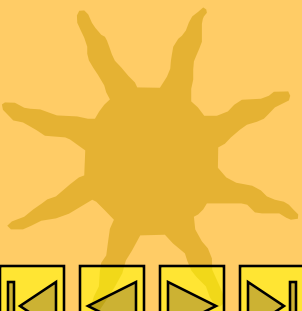




Further Reading

★ Readings and References

- Section 2.1 Character Strings
- Section 3.2 The String Class





StringBuilder

- ★ Strings are constants or String objects are **immutable**. In other words, their values cannot be changed after they are created.
- ★ In contrast, Java provides StringBuilder class which enables StringBuilder objects to be **mutable**.



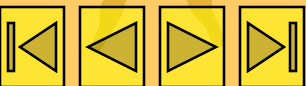


StringBuilder

```
String yourName = new String("Mike");  
String yourNameNew;
```

```
yourNameNew = yourName.concat("FUNG"); // yourName + "FUNG"  
System.out.println("Your name is: " + yourName);  
System.out.println("Your new name is: " + yourNameNew);
```

```
StringBuilder myName = new StringBuilder("Michael");  
// append the specified string to this character sequence.  
myName.append("FUNG");  
System.out.println(myName);
```





StringBuilder

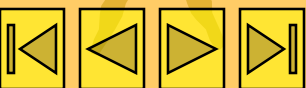
- ★ In Java, **String.concat()** instance method, **String operator "+"** and **String operator "+="** creates a **new String** object as a result while **StringBuilder.append()** will not.
- ★ Therefore, the following timing experiments with String is a lot **slower** than that with StringBuilder.



StringBuilder

★ StringBuilder is more **efficient** than String.

```
public static void testString () {  
  
    String s = "";  
    int repeat = 50000;  
  
    long begin = System.currentTimeMillis();  
  
    for (int i = 0; i < repeat; i++)  
        s += "java";  
  
    long elapsed = System.currentTimeMillis() - begin;  
  
    System.out.println("The time for "  
                        + s.getClass().getName()  
                        + " is: " + elapsed  
                        + " millisecond");  
}
```





StringBuilder

```
run:  
The time for java.lang.StringBuilder is: 0 millisecond  
The time for java.lang.String 12422 millisecond  
成功构建 (总时间: 12 秒)
```

★ StringBuilder is more **efficient** than String.

```
public static void testStringBuilder () {  
  
    StringBuilder sb = new StringBuilder();  
    int repeat = 50000;  
  
    long begin = System.currentTimeMillis();  
  
    for (int i = 0; i < repeat; i++)  
        sb.append("java");  
  
    long elapsed = System.currentTimeMillis() - begin;  
  
    System.out.println("The time for "  
                        + sb.getClass().getName()  
                        + " is: " + elapsed  
                        + " millisecond");  
}
```

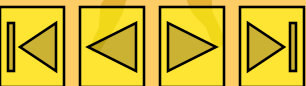
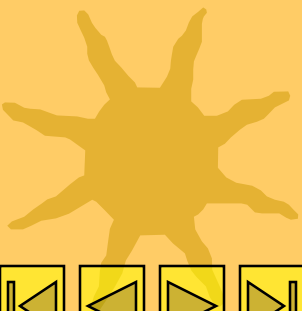




Exercise 1

★ Write a program that computes your initials from your full name and displays them.

e.g. Input: Michael Fung
Output: M.F.



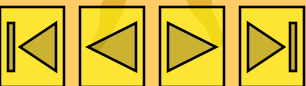


Exercise 2

- ★ Write a program that replaces all 't's with 'l's in a string where 't' is not at the beginning or at the end of the string.
e.g.

Input: toyhettowordthat

Output: toyhelloworldthat





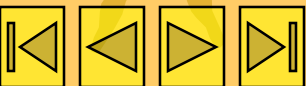
Exercise 3

- ★ Write a program that firstly **separates** a name to several parts around the match of white space like ' ', then converts the first letter of each part from **lowercase** to **uppercase** and finally outputs all parts.

e.g.

Input: jackie smith

Output: Jackie
Smith





Exercise 4

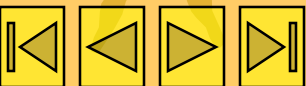
- ★ Write a program that **removes leading and trailing white spaces**, and then finds the **number of words**, **number of letters** as well as the **index** of the **third** word in the resulting string, all without white space.

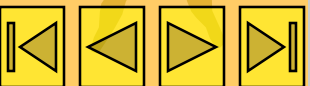
e.g. Input: " I,love,Java,a,lot "

Output: 5, 13, 7

i.e. Trim to "I,love,Java,a,lot"

→ 5 words, 13 letters, 'J' of 'Java' is at index 7





END