



IECS273/274 [1111 3670/3671] 物件導向設計與實習

09#1 Mutable and Immutable Objects in Java





01

► Mutable and Immutable Objects

▶ Mutable and Immutable Objects

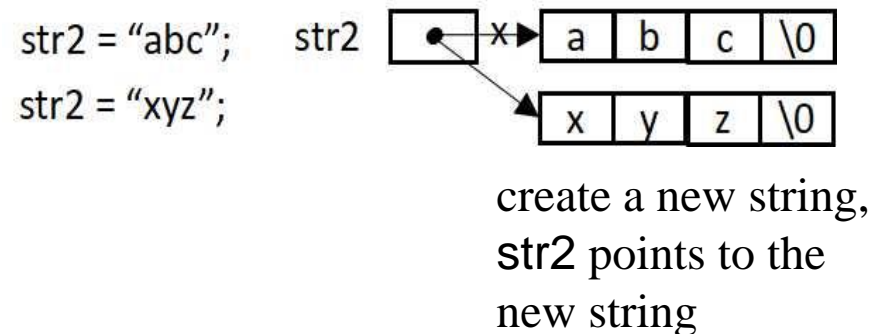
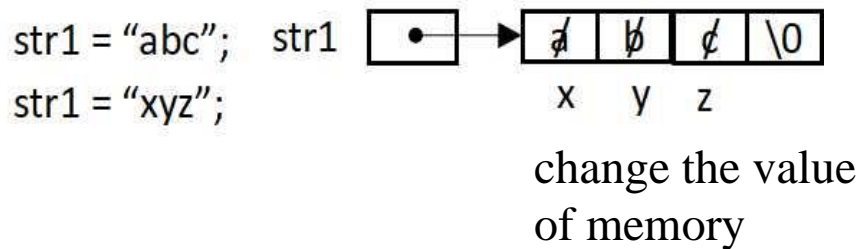
- Object-oriented programming languages have two different kinds of data: **mutable objects** and **immutable objects**.

- The *memory state* of **mutable objects** *can be changed*.

Assuming str1 is a string variable of a mutable object, consider the assignment statements: str1 = "abc"; str1 = "xyz";

- The *memory state* of **immutable objects** *cannot be changed*.

Assuming str2 is a string variable of an immutable object, consider the assignment statement: str2 = "abc"; str2 = "xyz";





02 ▶ Integer

► Integer in Java

- The **Integer** class built into the Java programming language is an *immutable* object with a value in the range -128 and 127, and is stored in a special memory area called the **constant pool**.

```
Integer i1 = 100; // Create object 100, store in constant pool, i1 points to this object.
```

```
Integer i2 = 100; // Search the constant pool to see if there is an object 100,  
                  // if so, i2 points to the object.
```

```
Integer i3 = 105; // No object 105 in the constant pool, then create object 105,  
                  // i3 points to this object.
```

```
i1 = 300; i2 = 300; // 300 is not in the range of Integer values, JVM will create two  
                  // objects of the same value, i1 and i2 point to these two  
                  // objects, respectively.
```

```
public class ImmutableObject {  
  
    public static void main(String[] args) {  
        // Integer objects are immutable; create object 100 in the constant pool,  
        // i1 points to this object.  
        Integer i1 = 100;  
        // Search the constant pool to see if there is an object 100,  
        // if so, i2 points to the object.  
        Integer i2 = 100;  
        // There is no object 105 in the constant pool,  
        // then create object 105, i3 points to this object.  
        Integer i3 = 105;  
  
        System.out.println("i1=" + i1 + ", i2=" + i2);  
        if (i1==i2) System.out.println("i1 and i2 are the same reference.");  
        else System.out.println("i1 and i2 are not the same reference.");  
        System.out.println("=====");  
    }  
}
```

```
i1 = 300; // Integer value between -128 and 127.  
// 300 is not in the range of Integer values, JVM will create two objects  
// of the same value, i1 and i2 point to these two objects, respectively.  
i2 = 300;  
  
System.out.println("i1=" + i1 + ", i2=" + i2);  
if (i1==i2) System.out.println("i1 and i2 are the same reference.");  
else System.out.println("i1 and i2 are not the same reference.");  
}  
}
```

ImmutableObject.class

i1=100, i2=100

i1 and i2 are the same reference.

=====

i1=300, i2=300

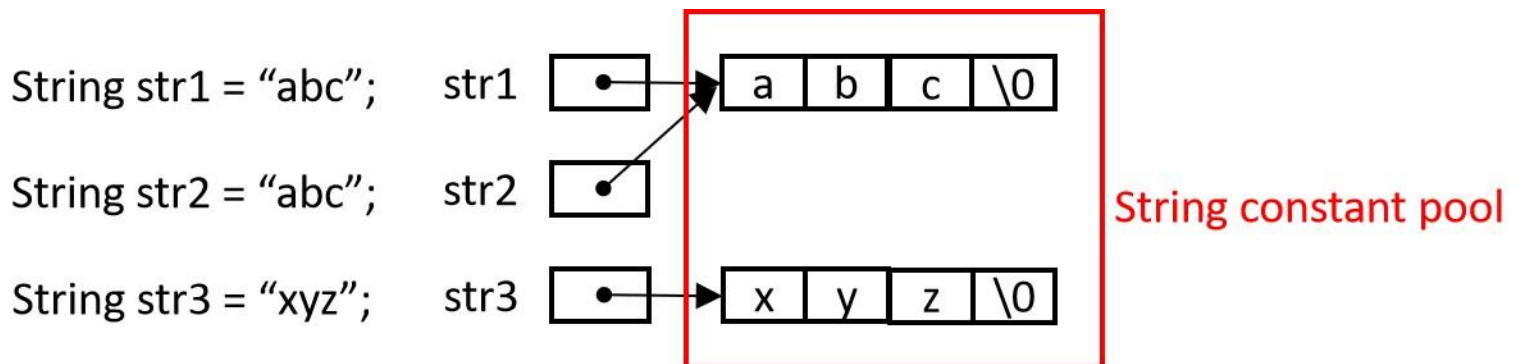
i1 and i2 are not the same reference.



03 ▶ String Class

String Class in Java

- Strings in Java are character arrays, with the difference that strings must be terminated with '\0'.
E.g.: `String str1 = "abc";`
Variable `str1` is a four-character array whose elements are `str1[0]='a'`, `str1[1]='b'`, `str1[2]='c'`, and `str1[3]='\0'`.
- A string in Java is an *immutable object*. When a string object is created directly without a constructor, the JVM will store the created characters in the **string constant pool**. When creating another string, the JVM first searches the string constant pool to see if the string exists, it points to the character array; otherwise, creates a new string constant. Object variable reference string constant in the string constant pool.



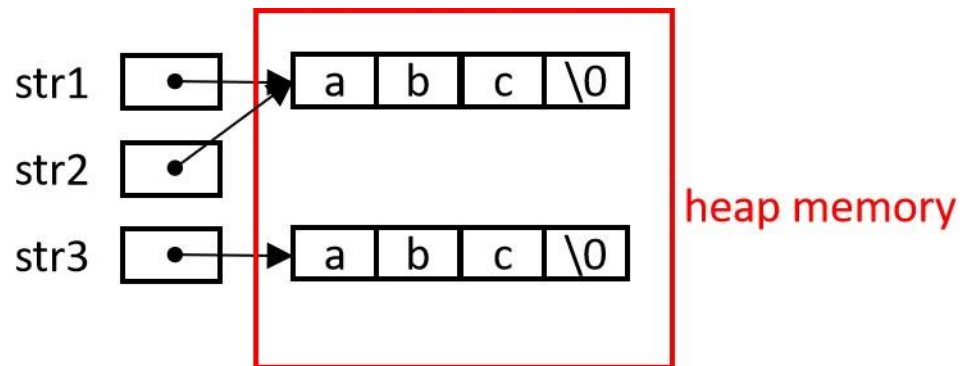
▶ String Class in Java

- Another way to create a string object in Java is to use a constructor, for example: `String str1 = new String("abc");`
 - The JVM will store the created string in *heap memory*.
 - When another string object with the same content is created, the JVM will create a new string and store it in heap memory.
 - In fact, the string constant pool is a special memory area of the heap memory.

```
String str1 = new String("abc");
```

```
String str2 = str1;
```

```
String str3 = new String("abc");
```



```
public class StringHeapAndPool {  
  
    public static void main(String[] args) {  
        String str1 = new String("abc");  
        String str2 = str1;  
        String str3 = new String("abc");  
        String str4 = "abc";  
        String str5 = "abc";  
        String str6 = str4;  
  
        if (str1==str3) System.out.println("Strings str1 and str3 are equal. (==)");  
        else System.out.println("Strings str1 and str3 are NOT equal. (==)");  
  
        if (str1.compareTo(str3)==0) System.out.println("Strings str1 and str3 are equal. (compareTo())");  
        else System.out.println("Strings str1 and str3 are NOT equal. (compareTo())");  
        System.out.println("=====");  
  
        if (str1==str4) System.out.println("Strings str1 and str4 are equal. (==)");  
        else System.out.println("Strings str1 and str4 are NOT equal. (==)");  
  
        if (str1.compareTo(str4)==0) System.out.println("Strings str1 and str4 are equal. (compareTo())");
```

```
else System.out.println("Strings str1 and str4 are NOT equal. (compareTo())");  
System.out.println("=====");
```

```
if (str4==str5) System.out.println("Strings str4 and str5 are equal. (==)");  
else System.out.println("Strings str4 and str5 are NOT equal. (==)");
```

```
if (str4.compareTo(str5)==0) System.out.println("Strings str4 and str5 are equal. (compareTo())");  
else System.out.println("Strings str4 and str5 are NOT equal. (compareTo())");  
System.out.println("=====");
```

```
if (str1==str2) System.out.println("Strings str1 and str2 are equal. (==)");  
else System.out.println("Strings str1 and str2 are NOT equal. (==)");
```

```
if (str2==str3) System.out.println("Strings str2 and str3 are equal. (==)");  
else System.out.println("Strings str2 and str3 are NOT equal. (==)");
```

```
if (str2==str4) System.out.println("Strings str2 and str4 are equal. (==)");  
else System.out.println("Strings str2 and str4 are NOT equal. (==)");
```

```
if (str1==str6) System.out.println("Strings str1 and str6 are equal. (==)");  
else System.out.println("Strings str1 and str6 are NOT equal. (==)");
```

StringHeapAndPool.java

```
if (str4==str6) System.out.println("Strings str4 and str6 are equal. (==)");  
else System.out.println("Strings str4 and str6 are NOT equal. (==)");  
  
if (str5==str6) System.out.println("Strings str5 and str6 are equal. (==)");  
else System.out.println("Strings str5 and str6 are NOT equal. (==)");  
}  
}
```

StringHeapAndPool.class

Strings str1 and str3 are NOT equal. (==)
Strings str1 and str3 are equal. (compareTo())

=====

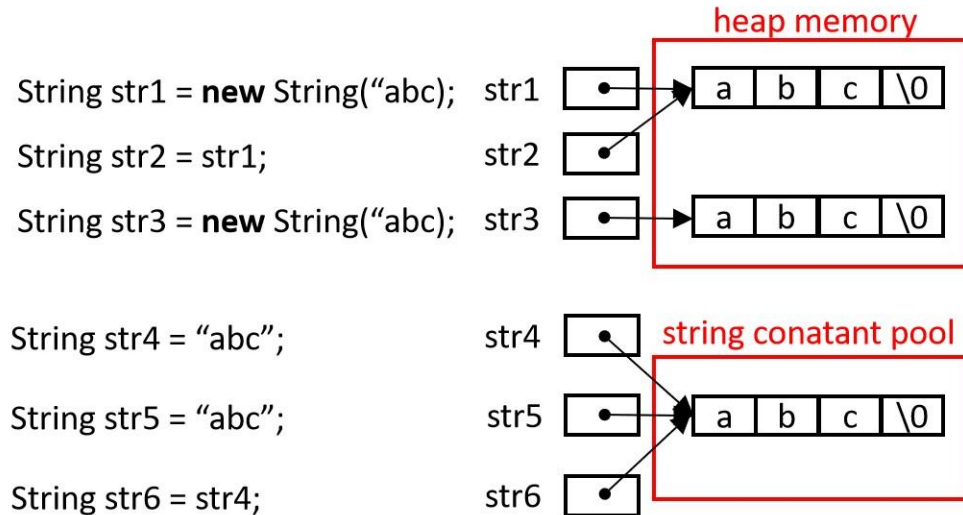
Strings str1 and str4 are NOT equal. (==)
Strings str1 and str4 are equal. (compareTo())

=====

Strings str4 and str5 are equal. (==)
Strings str4 and str5 are equal. (compareTo())

=====

Strings str1 and str2 are equal. (==)
Strings str2 and str3 are NOT equal. (==)
Strings str2 and str4 are NOT equal. (==)
Strings str1 and str6 are NOT equal. (==)
Strings str4 and str6 are equal. (==)
Strings str5 and str6 are equal. (==)



▶ String Methods in Java

■ Common methods of the **String** class:

- **int** length(): the length of a string.
- **char** charAt(**int** index): get a character at a given index.
- **void** getChars(**int** sourceStart, **int** sourceEnd, **char** target[], **int** targetStart): get multiple characters.
sourceStart specifies the subscript of the character at the beginning of the substring, and sourceEnd specifies the subscript of the next character after the end of the substring. Therefore, the substring contains the characters from sourceStart to sourceEnd-1. The array of received characters is specified by target, and the subscript value of the substring to start copying in target is targetStart.
- **char[]** toCharArray(): convert a string to a character array.
- **boolean** equals(String S): compares the contents of two string objects for equality.
"==" : compares the references of two string objects for equality.

► String Methods in Java

- **int** `compareTo(String)`: compare two strings, e.g.: `str1.compareTo(str2)`.
Returns 0 if `str1` and `str2` have the same content.
If the contents of `str1` and `str2` are not the same, return the difference of the codes of the first different characters.
Returns -1 if `str1` is a substring of `str2`.
Returns 1 if `str2` is a substring of `str1`.
- **int** `indexOf()` and **int** `lastIndexOf()`: index queries.
`indexOf()` queries the index of the first occurrence of a character or substring; returns -1 if the character or substring does not exist.
`lastIndexOf()` queries the index of the last occurrence of a character or substring; if the character or substring does not exist, return -1.
- `String substring()`: Intercept substring, there are two forms:
`String substring(int n)`: substring starting at index `n`.
`String substring(int m, int n)`: the substring starting at index `m` and before `n`.
- `String concat(String)`: Concatenates two strings; `append()` cannot be used.

► String Methods in Java

- `replace()`: replace character or string.
 String `replace(char oldChar, char newChar)`: Replace the character `oldChar` in the string with `newChar`.
 String `replace(String oldStr, String newStr)`: Replace the substring `oldStr` in the string with `newStr`.
- String `toLowerCase()` and String `toUpperCase()` convert to lowercase and uppercase.
- `trim()`: remove leading and trailing whitespace.
- `valueOf()`: Convert the parameter content to a string.
 static String `valueOf(Boolean)`: Boolean value
 static String `valueOf(char)`: character value
 static String `valueOf(int)`: integer value
 static String `valueOf(double)`: float value
 ...

```
public class StringMethods {  
  
    public static void main(String[] args) {  
        String str1 = new String("FengChia University");  
  
        System.out.println("String str1: " + str1);  
        System.out.println("Length of str1 str1.length(): " + str1.length());  
        System.out.println("=====");  
  
        System.out.println("String str1: " + str1);  
        System.out.println("Get the character at index 3 of str1, str1.charAt(3) " + str1.charAt(3));  
        char buff1[] = new char[20];  
        buff1[0] = '*';  
        buff1[1] = '@';  
        buff1[19] = '#';  
        System.out.print("character array buff1[: ]: ");  
        System.out.println(buff1);  
        str1.getChars(3, 7, buff1, 1);  
        System.out.print("Get the string at str1 of indices 3 to 6, str1.getChars(3, 7, buff1, 1): ");  
        System.out.println(buff1);  
    }  
}
```

StringMethods.java

```
char buff2[] = str1.toCharArray();
System.out.print("Convert the string str1 to a character array buff2[]=str1.toCharArray() : ");
System.out.println(buff2);
System.out.println("=====");

String str2 = new String(" Minjiang University");
System.out.println("String str2: " + str2);
if (str1==str2) System.out.println("Strings str1 and str2 are equal (str1==str2 returns true).");
else System.out.println("Strings str1 and str2 are not equal (str1==str2 returns false).");

if (str1.equals(str2)) System.out.println("Strings str1 and str2 are equal (str1.equals(str2) returns true).");
else System.out.println("Strings str1 and str2 are not equal (str1.equals(str2) returns false).");
System.out.println("Compare strings str1 and str2 (str1.compareTo(str2)): " + str1.compareTo(str2));
System.out.println("=====");

System.out.println("Compare strings \"abc\" and \"ABC\" (" + str1.compareTo(str2) + "): " + str1.compareTo(str2));
System.out.println("Compare strings \"ABC\" and \"abc\" (" + str2.compareTo(str1) + "): " + str2.compareTo(str1));
System.out.println("Compare strings \"00abc\" and \"00BCD\" (" + str1.compareTo(str2) + "): " +
    str1.compareTo(str2));
System.out.println("Compare strings \"AB\" and \"ABC\" (" + str1.compareTo(str2) + "): " + str1.compareTo(str2));
System.out.println("Compare strings \"ABCD\" and \"ABC\" (" + str1.compareTo(str2) + "): " +
    str1.compareTo(str2));
System.out.println("=====");
```

```
System.out.println("\n\' first occurrence at position " + str1.indexOf('n') + " of str1.");
System.out.println("\n\' last occurrence at position " + str1.lastIndexOf('n') + " of str1.");
System.out.println("\x\' first occurrence at position " + str1.indexOf('x') + " of str1.");
System.out.println("\x\' last occurrence at position " + str1.lastIndexOf('x') + " of str1.");
System.out.println("\jiang\' first occurrence at position " + str1.indexOf("Chia") + " of str1.");
System.out.println("\ver\' last occurrence at position " + str1.lastIndexOf("ver") + " of str1.");
System.out.println("\abc\' first occurrence at position " + str1.indexOf("abc") + " of str1.");
System.out.println("\abc\' last occurrence at position " + str1.lastIndexOf("abc") + " of str1.");
System.out.println("=====");

System.out.println("Substring starting at character 9 of str1: " + str1.substring(9));
System.out.println("The substring before the 9th to 12th characters of str1: " + str1.substring(9, 12));
System.out.println("=====");

System.out.println("concatenate two strings (\\"ABC\\".concat(\\"XYZ\\")): " + "ABC".concat("XYZ"));
System.out.println("concatenate two strings (cannot do \\"ABC\\".append(\\"XYZ\\")).");
System.out.println("=====");

System.out.println("Replace 'i' with 'a' in str1: " + str1.replace('i', 'a'));
System.out.println("The string str1 is unchanged: " + str1);
System.out.println("=====");
```

```
System.out.println("Convert all characters of str1 to lowercase: " + (str1+" 0123").toLowerCase());
System.out.println("The string str1 is unchanged: " + str1);
System.out.println("Convert all characters of str1 to uppercase: " + (str1+" 0123").toUpperCase());
System.out.println("The string str1 is unchanged:" + str1);
System.out.println("=====");
```

```
String str3 = "  FengChia University  ";
System.out.println("String str3: " + str3);
System.out.println("Remove leading and trailing spaces from the string str3: " + str3.trim());
System.out.println("=====");
```

```
int i1 = 1000;
float f1 = (float) 0.4562;
double d1 = Math.PI;
System.out.println("String.valueOf(true) : " + String.valueOf(true));
System.out.println("String.valueOf(str1==str2) : " + String.valueOf(str1==str2));
System.out.println("String.valueOf(str1) : " + String.valueOf(str1));
System.out.println("String.valueOf(buff1) : " + String.valueOf(buff1));
System.out.println("String.valueOf(i1) : " + String.valueOf(i1));
System.out.println("String.valueOf(f1) : " + String.valueOf(f1));
System.out.println("String.valueOf(d1) : " + String.valueOf(d1));
```

```
}
}
```

StringMethods.class

String str1: FengChia University

Length of str1 str1.length(): 甲

=====

String str1: FengChia University

Get the character at index 3 of str1 str1.charAt(3): 乙

character array buff1[]: 丙

Get the string at str1 of indices 3 to 6, str1.getChars(3, 7, buff1, 1): 丁

Convert the string str1 to a character array buff2[]=str1.toCharArray() : FengChia University

=====

String str2: Minjiang University

Strings str1 and str2 are not equal (str1==str2 returns false).

Strings str1 and str2 are not equal (str1.equals(str2) returns false).

Compare strings str1 and str2 (str1.compareTo(str2)): 戊

=====

Compare strings "abc" and "ABC" ("abc".compareTo("ABC")): 32

Compare strings "ABC" and "abc" ("ABC".compareTo("abc")): -32

Compare strings "00abc" and "00BCD" ("00abc".compareTo("00BCD")): 己

Compare strings "AB" and "ABC" ("AB".compareTo("ABC")): -1

Compare strings "ABCD" and "ABC" ("ABCD".compareTo("ABC")): 庚

=====

'n' first occurrence at position 2 of str1.

'n' last occurrence at position 10 of str1.

'x' first occurrence at position -1 of str1.

'x' last occurrence at position -1 of str1.

StringMethods.class

"jiang" first occurrence at position -1 of str1.

"ver" last occurrence at position 12 of str1.

"abc" first occurrence at position -1 of str1.

"abc" last occurrence at position -1 of str1.

=====

Substring starting at character 9 of str1: University

The substring before the 9th to 12th characters of str1: 辛

=====

concatenate two strings ("ABC".concat("XYZ")): ABCXYZ

concatenate two strings (cannot do "ABC".append("XYZ")) °

=====

Replace 'i' with 'a' in str1: 壬

The string str1 is unchanged: FengChia University

=====

Convert all characters of str1 to lowercase: fengchia university 0123

The string str1 is unchanged: FengChia University

Convert all characters of str1 to uppercase: FENGCHIA UNIVERSITY 0123

The string str1 is unchanged: FengChia University

=====

String str3: FengChia University

Remove leading and trailing spaces from the string str3: 癸

=====

StringMethods.class

```
String.valueOf(true) : true  
String.valueOf(str1==str2) : false  
String.valueOf(str1) : FengChia University  
String.valueOf(buff1) : *gChi      #  
String.valueOf(i1) : 1000  
String.valueOf(f1) : 0.4562  
String.valueOf(d1) : 3.141592653589793
```



04 ▶ StringBuffer Class

► StringBuffer Class in Java

- Java has another kind of string called **StringBuffer** class.
 - Objects of **StringBuffer** class are *mutable, writable strings*.
 - Strings of **StringBuffer** objects are stored in *heap memory*.
 - Most of the methods for changing strings are applied to **StringBuffer** objects.
 - When the content of a **StringBuffer** object is changed, its reference will also have side effects and be changed at the same time.

StringAppend.java

```
public class StringAppend {  
    public static void main(String[] args) {  
        String str1 = new String("ABC");  
        String str2 = new String("XYZ");  
  
        System.out.println("Not use append operation: ");  
        System.out.println("str1=" + str1);  
        // System.out.println("str1+str2" + str1.append(str2));  
        System.out.println("str1+str2=" + str1 + str2); // Cannot use append operation.  
        System.out.println("=====");  
  
        StringBuffer str3 = new StringBuffer("ABC");  
        StringBuffer str4 = str3;  
        str3.append(str2); // String append operation, str3 is modified.  
        System.out.println("Use the append operation on str3, and str4 references str3:");  
        System.out.println("str3=" + str3);  
        System.out.println("str4=" + str4);  
        System.out.println("=====");  
    }  
}
```

StringAppend.java

```
str4.append("RST");
System.out.println("Using the append operation on str4, str3 is also changed;");
System.out.println("str3=" + str3);
System.out.println("str4=" + str4);
System.out.println("=====");

System.out.println("Convert a String object to a StringBuffer object using the constructor;");
StringBuffer str5 = new StringBuffer(str1);
System.out.println("str5=" + str5);
System.out.println("Convert the String object to a StringBuffer object using append();");
StringBuffer str6 = new StringBuffer();
str6.append(str2);
System.out.println("str6=" + str6);
String str7;
System.out.println("Convert the StringBuffer object to a String object using toString();");
str7 = str3.toString();
System.out.println("str7=" + str7);
}
}
```

StringAppend.class

Not use append operation:

```
str1=ABC
```

```
str1+str2=ABCXYZ
```

```
=====
```

Use the append operation on str3, and str4 references str3:

```
str3=ABCXYZ
```

```
str4=ABCXYZ
```

```
=====
```

Using the append operation on str4, str3 is also changed;

```
str3=ABCXYZRST
```

```
str4=ABCXYZRST
```

```
=====
```

Convert a String object to a StringBuffer object using the constructor;

```
str5=ABC
```

Convert the String object to a StringBuffer object using append();

```
str6=XYZ
```

Convert the StringBuffer object to a String object using toString();

```
str7=ABCXYZRST
```

► StringBuffer Methods in Java

- StringBuffer defines the following constructors:

`StringBuffer();` // Default constructor

`StringBuffer(int size);` // Parameters size: buffer size

`StringBuffer(String str);` // Parameters str: string

`StringBuffer(CharSequence chars);` // Parameters chars: character sequence

- **int** `length()`: returns string length
- **int** `capacity()`: returns the allocated space for the string buffer
- `StringBuffer append(String s)`: concatenate two strings
- `StringBuffer insert(int offset, String str)`: insert string str from offset position
- `StringBuffer replace(int startIndex, int endIndex, String str)`: replace the string from startIndex to endIndex-1 with str

► StringBuffer Methods in Java

- `StringBuffer delete(int startIndex, int endIndex)`: delete the string from startIndex to endIndex-1
- `StringBuffer reverse()`: reverse the string
- **void** `ensureCapacity(int minimumCapacity)`: ensures buffer allocation space is at least equal to the given value minimumCapacity
- **char** `charAt(int index)`: returns the character at the specified position
- `String substring(int beginIndex)`: returns a substring from the specified beginIndex
- `String substring(int beginIndex, int endIndex)`: returns the substring between beginIndex and endIndex-1


```
public class StringBufferMethods {  
  
    public static void main(String[] args) {  
        CharSequence charSeq = "ABCDE";  
        StringBuffer str1 = new StringBuffer();  
        StringBuffer str2 = new StringBuffer(100);  
        StringBuffer str3 = new StringBuffer("abcde");  
        StringBuffer str4 = new StringBuffer(charSeq);  
  
        System.out.println("str1= " + str1);  
        System.out.println("str2= " + str2);  
        System.out.println("str3= " + str3);  
        System.out.println("str4= " + str4);  
  
        System.out.println("str1.lenght()= " + str1.length());  
        System.out.println("str1.capacity()= " + str1.capacity());  
  
        System.out.println("str2.lenght()= " + str2.length());  
        System.out.println("str2.capacity()= " + str2.capacity());  
  
        System.out.println("str3.lenght()= " + str3.length());
```

StringBufferMethods.java

```
System.out.println("str3.capacity()= " + str3.capacity());

System.out.println("str4.lenght()= " + str4.length());
System.out.println("str4.capacity()= " + str4.capacity());
System.out.println("=====");

System.out.println("str4.replace(1, 3, \"UVWXYZ\")= " + str4.replace(1, 3, "UVWXYZ"));
System.out.println("str4.delte(1, 3)= " + str4.delete(1, 3));
System.out.println("str3.reverse()= " + str3.reverse());
System.out.println("str4.reverse()= " + str4.reverse());
System.out.println("=====");

str4.ensureCapacity(50);
System.out.println("After str4.ensureCapacity(50), str4.capacity()= " + str4.capacity());
System.out.println("str4.charAt(3)= " + str4.charAt(3));
System.out.println("str4.substring(3)= " + str4.substring(3));
System.out.println("str4.substring(3, 6)= " + str4.substring(3, 6));
}
```

StringBufferMethods.class

```
str1=  
str2=  
str3= abcde  
str4= ABCDE  
str1.length()= 0  
str1.capacity()= 16  
str2.length()= 0  
str2.capacity()= 100  
str3.length()= 5  
str3.capacity()= 21  
str4.length()= 5  
str4.capacity()= 21  
=====  
str4.replace(1, 3, "UVWXYZ")= AUVWXYZDE  
str4.delete(1, 3)= AWXYZDE  
str3.reverse()= edcba  
str4.reverse()= EDZYXWA  
=====  
After str4.ensureCapacity(50), str4.capacity()= 50  
str4.charAt(3)= Y  
str4.substring(3)= YXWA  
str4.substring(3, 6)= YXW
```

► Difference between String and StringBuffer

- Strings are represented in Java using the **String** class.
 - But the **String** class representing strings has one of the biggest problems: "A string constant once declared cannot be changed, while a **String** object can be changed, but what changes is the memory address it points to."
 - Therefore, the **String** class is not suitable for frequently modified string operations; therefore, in this case, the **StringBuffer** class can often be used, that is, the **StringBuffer** class is convenient for users to modify content.
 - Use "+" as the data concatenation operation in the **String** class; and use append() method in the **StringBuffer** class to concatenate strings.
 - **String** and **StringBuffer** are defined differently:
public final class String extends Object implements Serializable, Comparable<String>, CharSequence, Constable, ConstantDesc
public final class StringBuffer extends Object implements Serializable, CharSequence
 - Both are subclasses of **CharSequence** interface, that is, objects of **String** class and **StringBuffer** class can be instantiated into the **CharSequence** interface using automatic up casting.



05 ▶ Programming Practice

► Practice 1 : Cipher Algorithms

A cryptographic technique is the multi-letter substitution cipher. A multi-letter substitution cipher is called Vigenère squares with keywords is used to encode English alphabetic text. The first row of the Vigenère square is an alphabetical sequence of 26 English letters, and then each row below is a cyclic left rotation of the row above it. The **Vigenère square** is shown as the table below:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	N	N	O	P	Q	R	S	T	U	V	W	X	Y

Practice 1 : Cipher Algorithms

In fact, the Vigenère square can be thought of as 26 code books indexed by the letters of the first column. A **keyword** is any English word, and it will repeat itself in a constant loop to the same length as the original text. The selection of the code book is to align the repeated keywords with the English letters of the original text, select the keyword letters in the same position, and then match the first letter of the code book to select the code book. Using the selected code book, the original letters will be converted into encrypted letters. Then, move to the next letter of the original text, and repeat the selection of the code book and encrypted letters until all the letters of the original text are translated.

For example, if the keyword is "FENGCHIA", and the original text is "OBJECT ORIENTED PROGRAMMING AND JAVA", the encoded text is "TFWKEA WRNIAZGK XRTKEGOTQNL EAJ LHDA".

Keyword	F	E	N	G	C	H		I	A	F	E	N	G	C	H		I	A	F	E	N	G	C	H	I	A
Original text	O	B	J	E	C	T		O	R	I	E	N	T	E	D		P	R	O	G	R	A	M	M	I	N
Encoded text	T	F	W	K	E	A		W	R	N	I	A	Z	G	K		X	R	T	K	E	G	O	T	Q	N
Keyword	F		E	N	G		C	H	I	A																
Original text	G		A	N	D		J	A	V	A																
Encoded text	L		E	A	J		L	H	D	A																

► Practice 1 : Cipher Algorithms

Using the coding technology of Vigenère square cipher, and selecting "FENGCHIA" as the keyword, write a Java program to

- 1) input a string of English letters,
- 2) convert lowercase letters to uppercase,
- 3) encode it into the encoded text and print the encoded text,
- 4) Decode the encoded text into the original text and print the result text.



Keyword: FENGCHIA

Original Text: **Object Oriented Programming**

Uppercase Text: OBJECT ORIENTED PROGRAMMING

Encoded text: TFWKEA WRNIAZGK XRTKEGOTQNL

Decoded text: OBJECT ORIENTED PROGRAMMING



06 ▶ Assignment 4

► Assignment 4 : Vigenère Square with Substitution Cipher

A substitutional Vigenère square is a variance of Vigenère square combining with substitution cipher. The first row is a substitution cipher code book and each of the following row is the cyclic left rotation of the row right on the top of it. An example of a substitutional Vigenère square is shown as the following:

Alphabet letters	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encode book	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T
Left rotation of the line above	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I
	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C
	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O
	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X
	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D
	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P
	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B
	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U
	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q
	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J
	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A
	H	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R
	K	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H
	N	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K
	G	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N
	W	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G
	Y	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W
	L	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y
	E	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L
	Z	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E
	S	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z
	V	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S
	F	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V
	M	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F
	T	I	C	O	X	D	P	B	U	Q	J	A	R	H	K	N	G	W	Y	L	E	Z	S	V	F	M

► Assignment 4 : Vigenère Square with Substitution Cipher


A keyword can be any English word, it will be repeatedly concatenate itself until the same length as the encoded text. The encoded text and the repeated keyword is aligned and then a code to book is selected. Selection of the code book is to match the aligned letter of the keyword with the first letter of the code book. With the selected code book, the letter of the encoded text is then translated to a ciphered letter. For example, if the keyword is "FENGCHIA" and the text is "Object Oriented Programming and Java", the encoding of the text is shown as below:

Keyword	F	E	N	G	C	H		I	A	F	E	N	G	C	H		I	A	F	E	N	G	C	H	I	A
Original text	O	B	J	E	C	T		O	R	I	E	N	T	E	D		P	R	O	G	R	A	M	M	I	N
Encoded text	R	Z	F	E	X	P		N	C	P	F	C	Q	P	G		G	C	R	T	P	G	K	M	Q	F
Keyword	F		E	N	G		C	H	I	A																
Original text	G		A	N	D		J	A	V	A																
Encoded text	X		E	C	L		A	H	S	A																

► Assignment 4 : Vigenère Square with Substitution Cipher

The encoded text is "RZFEXP NCPFCQPG GCRTPGKMQFX ECL AHSA". Write a Java program to perform the following steps:

- (1) Input an English text;
- (2) Convert all lower case letters to upper case letters;
- (3) Encode the text using the Vigenère square with the code book given in the substitutional Vigenère square;
- (4) Output the original text and the encoded text;
- (5) Decode the encoded text and output the decoding result.



<terminated> SubstitutionalVigenereSquareCipher [Java Application] C:\Program File

Keyword: FENGCHIA

Original text: Object Oriented Programming and Java

Upper case text: OBJECT ORIENTED PROGRAMMING AND JAVA

Encoded text: RZFEXP NCPFCQPG GCRTPGKMQFX ECL AHSA

Decoded text: OBJECT ORIENTED PROGRAMMING AND JAVA