The String Class

Strings are used for storing text.

A String variable contains a collection of characters surrounded by double quotes:

String is a sequence of characters.

In java, objects of String are immutable which means a constant and cannot be changed once created.

Creating a String

- There are two ways to create a string in Java:
  - ***String literal***

```
String s = “hello”;
```

- **Using *new* keyword**

```
String s = new String (“hello”);
```

String class facts

- An object of the String class represents a string of characters.
- The String class belongs to the java.lang package, which does not require an import statement.
- Like other classes, String has constructors and methods.
- Unlike other classes, String has two operators, + and += (used for concatenation).

Literal Strings

- are anonymous objects of the String class
- are defined by enclosing text in double quotes. “This is a literal String”
- don’t have to be constructed.
- can be assigned to String variables.
- can be passed to methods and constructors as parameters.
- have methods you can call.

Example 1

```
class StringStorage {
  public static void main(String args[])
  {
    String s1 = "TAT";
    String s2 = "TAT";
    String s3 = new String("TAT");
    String s4 = new String("TAT");
    System.out.println(s1);
    System.out.println(s2);
    System.out.println(s3);
    System.out.println(s4);
  }
}
```

Example 2 Adding Numbers

- String x = "10";
- String y = "20";
- String z = x + y; // z will be 1020 (a String)

Literal String examples

```
//assign a literal to a String variable
String name = “Robert”;

//calling a method on a literal String
char firstInitial = “Robert”.charAt(0);

//calling a method on a String variable
char firstInitial = name.charAt(0);
```

Immutability

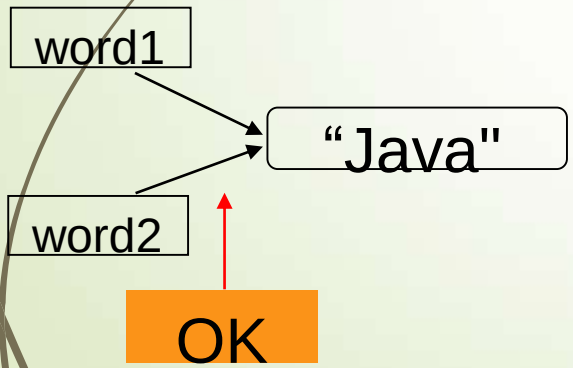
- Once created, a string cannot be changed: none of its methods changes the string.
- Such objects are called *immutable*.

- Immutable objects are convenient because several references can point to the same object safely: there is no danger of

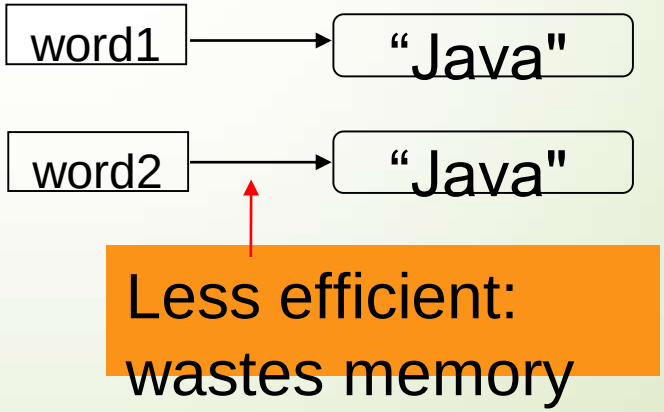
Advantages Of Immutability

Uses less memory.

```
String word1 = "Java";
String word2 = word1;
```



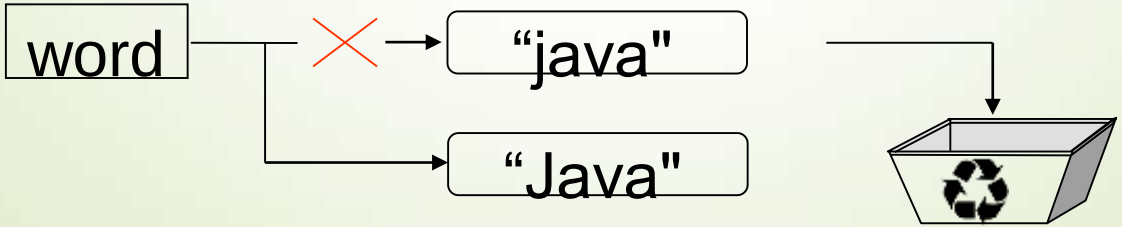

```
String word1 = “Java";
String word2 = new String(word1);
```



Disadvantages of Immutability

Less efficient — you need to create a new string and throw away the old one even for small changes.

```
String word = “Java";
char ch = Character.toUpperCase(word.charAt (0));
word =  ch + word.substring (1);
```



Empty Strings

- An empty String has no characters. It's length is 0

```
String word1 = "";
String word2 = new String();
```

Empty strings

```
private String errorMsg;
```

errorMsg is null

- Not the same as an uninitialized String.

No Argument Constructors

- No-argument constructor creates an empty String. Rarely used.

```
String empty = new String();
```

- A more common approach is to reassign the variable to an empty literal String. (Often done to reinitialize a variable used to store input.)

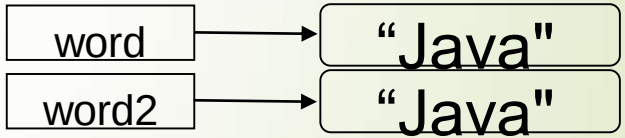
```
String empty = "";//nothing between quotes
```

Copy Constructors

- Copy constructor creates a copy of an existing String. Also rarely used.
- Not the same as an assignment.

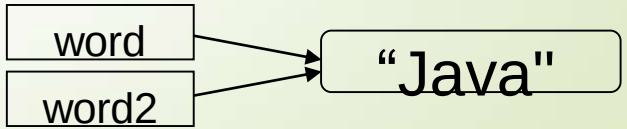
Copy Constructor: Each variable points to a different copy of the String.

```
String word = new String("Java");
String word2 = new String(word);
```



Assignment: Both variables point to the same String.

```
String word = "Java";
String word2 = word;
```



Other Constructors

Most other constructors take an array as a parameter to create a String.

```
char[] letters = {‘J’, ‘a’, ‘v’, ‘a’};
String word = new String(letters);//”Java”
```

Methods — length, charAt

int length();

- Returns the number of characters in the string

char charAt(i);

- Returns the char at position i.

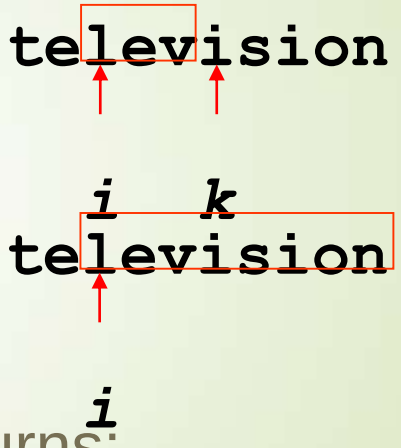
Character positions in strings are numbered starting from 0 – just like arrays.

	Returns:
”Problem".length();	7
”Window".charAt (2);	’n'

Methods — substring

Returns a new String by copying characters from an existing String.

- String subs = word.**substring** (i, k);
  - returns the substring of chars in positions from **i** to **k-1**
- String subs = word.**substring** (i);
  - returns the substring from the **i**-



	Returns:
”television".substring (2,5);	“lev"
“immutable".substring (2);	“mutable"
“bob".substring (9);	"" (empty string)

Methods — Concatenation

String word1 = “re”, word2 = “think”; word3 = “ing”;

int num = 2;

- String result = word1 **+** word2;

  //concatenates word1 and word2 “rethink“

- String result = word1.**concat** (word2);

  //the same as word1 + word2 “rethink“

- **result +=** word3;

  //concatenates word3 to result “rethinking”

- r**esult** += num; //converts num to String

Methods — Find (indexOf)

String name =“President George Washington";

	Returns:
date.indexOf (‘P');	0
date.indexOf (‘e');	2
date.indexOf (“George");	10
date.indexOf (‘e', 3);	6

(starts searching at position 3)

(not found)

Methods — Equality

boolean b = word1.**equals**(word2);

returns **true** if the string **word1** is equal to **word2**

boolean b = word1.**equalsIgnoreCase**(word2);

returns **true** if the string **word1** matches **word2**,

```
b = "Raiders".equals("Raiders");//true
b = "Raiders".equals("raiders");//false
b = "Raiders".equalsIgnoreCase("raiders");//true
```

```
if(team.equalsIgnoreCase("raiders"))
        System.out.println("Go You " + team);
```

Methods — Comparisons

int diff = word1.**compareTo**(word2);
returns the "difference" **word1 – word2**

int diff = word1.**compareToIgnoreCase**(word2);
returns the "difference" **word1 – word2**,
case-blind

Usually programmers don't care what the numerical "difference" of **word1 - word2** is, just whether the difference is negative (word1 comes before word2), zero (word1 and word2 are equal) or positive (word1 comes after word2). Often used in conditional statements.

```
if(word1.compareTo(word2) > 0){
        //word1 comes after word2…
}
```

Comparison Examples

```
//negative differences
diff = “apple”.compareTo(“berry”);//a before b
diff = “Zebra”.compareTo(“apple”);//Z before a
diff = “dig”.compareTo(“dug”);//i before u
```

```
//zero differences
diff = “apple”.compareTo(“apple”);//equal
diff = “dig”.compareToIgnoreCase(“DIG”);//equal
```

```
//positive differences
diff = “berry”.compareTo(“apple”);//b after a
diff = “apple”.compareTo(“Apple”);//a after A
diff = “BIT”.compareTo(“BIG”);//T after G
diff = “huge”.compareTo(“hug”);//huge is longer
```

Methods — trim

String word2 = **word1.trim** ();

returns a new string formed from **word1** by removing white space at both ends

does not affect whites space in the middle

```
String word1 = “ Hi Bob “;
String word2 = word1.trim();
//word2 is “Hi Bob” – no spaces on either end
//word1 is still “ Hi Bob “ – with spaces
```

Methods — replace

String word2 = word1.**replace**(oldCh, newCh);

returns a new string formed from **word1** by replacing all occurrences of **oldCh** with **newCh**

```
String word1 = “rare“;
String word2 = “rare“.replace(‘r’, ‘d’);
//word2 is “dade”, but word1 is still “rare“
```

Methods — Changing Case

String word2 = word1.**toUpperCase**();
String word3 = word1.**toLowerCase**();

returns a new string formed from **word1** by converting its characters to upper (lower)

```
String word1 = “HeLLo“;
String word2 = word1.toUpperCase();//”HELLO”
String word3 = word1.toLowerCase();//”hello”
//word1 is still “HeLLo“
```

Replacements

- Example: to “convert” word1 to upper
  reference.

```
word1 = word1.toUpperCase();
```

```
word1.toUpperCase();
```

word1 remains unchanged

- A common bug:

Numbers to Strings

Three ways to convert a number into a string:

1. s = “” + 123;//”123”
2. String s = Integer.toString (i);

   s = Integer.toString(123);//”123”
   s = Double.toString(3.14); //”3.14”

3. s = String.valueOf(123);//”123”

Integer and **Double** are “wrapper” classes from **java.lang** that represent numbers as objects. They also provide useful static methods.

- **How is it possible for two String objects with identical values not to be equal under the ==**
- **operator?**
- The == operator compares two objects to determine if they are the same object in memory.
- It is possible for two String objects to have the same value, but located indifferent areas of memory.

StringBuffer

- The StringBuffer class is used to represent characters that can be modified. This is simply used for concatenation or manipulation of the strings.

- *StringBuffer* is mainly used for the dynamic string concatenation which enhances the performance.
- A string buffer implements a mutable sequence of characters

- *append()*
- This is the append() function used for the concatenate the string in string buffer. This is better to use for dynamic string concatenation. This function works like a simple string concatenation such as : String str = str + "added string";.

- *insert()*
- This is the insert() function used to insert any string or character at the specified position in the given string.

- *reverse()*
- This is the reverse() function used to reverse the string present in string buffer.

- ***setCharAt()***
- This is the setCharAt() function which is used to set the specified character in buffered string at the specified position of the string in which you have to set the given character.

- ***charAt()***

- This is the charAt() function which is used to get the character at the specified position of the given string.

- ***substring()***
- This is the substring() function which is used to get the sub string from the buffered string from the initial position to end position (these are fixed by you in the program).
- ***deleteCharAt()***
- This is the deleteCharAt() function which is used to delete the specific character from the buffered string by mentioning that's position in the string.

- ***length()***
- This is the length() function is used to finding the length of the buffered string.
- ***delete()***
- This is the delete() function is used to delete multiple character at once from *n* position to *m* position (n and m are will be fixed by you.) in the buffered string.
- ***capacity()***
- This is the capacity() function is used to know about the current characters kept which is displayed like : number of characters + 6.

Example

```
StringBuffer strBuf1 = new StringBuffer("Bob");
StringBuffer strBuf2 = new StringBuffer(100); //With capacity 100

StringBuffer strBuf3 = new StringBuffer(); //Default Capacity 16

System.out.println("strBuf1 : " + strBuf1);

System.out.println("strBuf2 capacity : " + strBuf2.capacity());

System.out.println("strBuf3 capacity : " + strBuf3.capacity());
```

StringBuffer Capacity

- Every StringBuffer object has a capacity associated with it.
- The capacity of the StringBuffer is the number of characters it can hold.
- It increases automatically as more contents added to it.

For example,

- StringBuffer stringBuffer = new StringBuffer(“Hello World”);
- System.out.println(stringBuffer.capacity());

This will print 27 (11 + 16) on console when executed.

- The actual number of characters in StringBuffer can be obtained by following method.
- int length()

For example,

- String str = “Hello”;
- StringBuffer stringBuffer = new StringBuffer(str);
- Here, capacity of stringBuffer object would be 5 + 16 = 21.

Review Questions:

1. The String class is part of what package?
2. What does the String class have that other classes do not have?
3. "Text enclosed in quotes is called ?"
4. What is the returned value for "Rumplestiltskin".length()?
5. Define immutable objects.

Review (cont'd):

6. How does immutability of Strings make Java more efficient?
7. How do you declare an empty string?
8. "Bob" + " " + "Smith" is called ____ ?

Review (cont'd):

11. String city = "Bloomington";
    What is returned by city.charAt (2)?
12. By city.substring(2, 4)?
13. By city.lastIndexOf('o')?
14. By city.indexOf(3)?
15. What does the trim method do?

Review (cont'd):

16. "sam".equals("Sam") returns ?
17. What kind of value does "sam".compareTo("Sam") return?
18. What will be stored in s?
    s = "mint".replace('t', 'e');
19. What does s.toUpperCase() do to s?
20. Name a simple way to convert a number into a string.

Compare Strings Example

```
public class CompareToExample {
    public static void main(String args[]) {
        String str1 = "String method tutorial";
        String str2 = "compareTo method example";
        String str3 = "String method tutorial";
        int var1 = str1.compareTo( str2 );
        System.out.println("str1 & str2 comparison: "+var1);
        int var2 = str1.compareTo( str3 );
        System.out.println("str1 & str3 comparison: "+var2);
```

Example 3 CompareTo()

```
public class JavaExample {
    public static void main(String args[]) {
        //uppercase
        String str1 = "HELLO";
        //lowercase
        String str2 = "hello";;

        System.out.println(str1.compareTo(str2));
    }
}
```

Questions on String

- **What is String?**

  String is a class in java which is present in java.lang package. According to Oracle docs,
  The String class represents character strings. Strings are constant, their values can not be changed after they are created.

- **Is String immutable in java?**

  Yes, String class is immutable in java. Immutable means once the object is created, its value can not be changed.

  Q3 Is String a keyword in java ?

  No, String is not a keyword in java.

- **Is String primitive type or object (derived) type in java?**

  String is object(derived) type in java.

- **Can we use String in switch statement?**

  Yes, you can use String in switch statement in java 7. Prior to java 7 , you had to use if-else statements to achieve the task.

- How many objects will be created for the following code:

- String str1 = "abc";

- String str2 = "abc";

- Only one object is created. String str1 will create a

Example

```
public class MyStringInitialization {
    public static void main(String a[]){
        String abc = "This is a string object";
        String bcd = new String("this is also string object");
        char[] c = {'a','b','c','d'};
        String cdf = new String(c);
        String junk = abc+" This is another String object";
    }}
```

Lab

- Q1 wap to initialize string by assigning value in program and display it Q 2 wap to ask two string values from user and check if they are equal or not
- Q 3 wap to compare two strings
- Q 4 wap to print string character by character
- Q 5 wap to check if string is palindrome or not
- Q 6 wap to count number of vowels and consonants in string
- Q 7 wap to count number of words in given string
-