String Handling

- In Java, a string is a sequence of characters.
- Java implements strings as object of type String.

The String Constructors

- String s = new String( );
  - default constructor
  - create an empty string
- String s = new String(char chars[ ]);
  - create a string initialized by an array of characters
  - Example : char chars[ ] = {‘a, ‘b’, ‘c’, ‘d’. ‘e’};

    String s = new String(chars);

- String s = new String(char chars[ ], int startIndex, int numChars);
  - can specify a subrange of a character array as an initializer.
  - startIndex specifies the index at which the subrange begins.
  - numChars specifies the number of characters to use.
  - Example : char chars[ ] = {'a', 'b', 'c', 'd', 'e', 'f'}
    String s = new String(chars, 2, 3);
    -> initializes s with the characters 'cde'
- String s = new String(String strObj);
  - construct a String object that contains the same character sequence as another String object using this constructor.
  - Example: char c[ ] = {'J', 'A', 'V', 'A'};

```
String s1 = new String(c);
String s2 = new String(s1);
System.out.println(s1);
System.out.println(s2);
```

- String s = new String(byte asciiChars[ ]);
  - Constructors that initialize a string when given a byte array.
  - asciiChars specifies the array of bytes.
- String s = new String(byte asciiChars[ ],int startIndex, int numChars);
  - Example: byte ascii[ ] = {65, 66,67,68,69,70};

```
String s1 = new String(ascii);
System.out.println(s1);
String s2 = new String(ascii, 2,3);
System.out.println(s2);
```

Output:

ABCDEF

CDE

String Length

- The length of the string is the number of characters it contains.
- To obtain this value, call the length( ) method.

```
int length( )
```

Example

```
char chars[ ] = {'a', 'b', 'c'};
String s = new String(chars);
System.out.println(s.length( ));
```

Special String Operations

- automatic creation of new string instances from string literals.
- concatenation of multiple String objects by use of the + operator.
- conversion of other data types to a String representation.

String Literals

- For each string literal in your program, Java automatically constructs a String object.

  Example: String str = “abc”
- String literals can be used to call methods directly as if it were an object reference.

  Example: System.out.println(“abc”.length( ));

String Concatenation

- Java does not allow operators to be applied to String objects.
- One exception is, the + operator, which concatenates two strings producing a String object as the result.
- Example: String age = “9”;

```
String s = “He is” + age + “years old”;
System.out.println(s);
```

String Concatenation with other data types

- Strings can be concatenated with other types of data.
- Example: int age = 9;

```
String s = “He is” + age + “years old”;
System.out.println(s);
```

- Mixing other types of operations with String concatenation expressions

```
String s = “four:” +2+2;
System.out.println(s);          // output: four:22
```

- If 2 and 2 has to be added, then use the parenthesis

```
String s = “four:” +(2+2);
System.out.println(s);          // output: four:4
```

Character Extraction

- Many of the String methods employ an index into the String for their operation.
- Like arrays, the String indexes begin at zero.

charAt( )

- To extract a single character from a String.
- General form

    char charAt(int where)

    where -> index of the character; value must be non-negative.

Example: char ch;

    ch = “abc”.charAt(1); //assign the value “b” to ch

getChars( )

- To extract more than one character at a time.
- General form

void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)

sourceStart -> index of the beginning of the substring

sourceEnd -> index that is one past the end of the desired substring

(from sourceStart to sourceEnd – 1)

target[ ] -> array that receive the characters specified.

targetStart -> index within target at which the substring will be copied

- Example

```
class getCharsDemo
{
        public static void main(String args[ ])
        {
                String s = “This is a Demo program”;
                int start = 10, end =14;
                char buf[ ] = new char[10];
                s.getChars(start, end, buf, 0);
                System.out.println(buf);
        }
}
```

getBytes( )

- alternative to getChars( ) that stores the characters in an array of bytes.
- General form:

```
byte[ ] getBytes( )
```

toCharArray( )

- To convert all the characters in a String object into a character array, the easy way is to call toCharArray( )
- General form:

```
char[ ] toCharArray( )
```

String Comparison

❑ equals( ) and equalsIgnoreCase( )

- To compare two strings for equality

  boolean equals(String str)

- To perform a comparison that ignores case differences

  boolean equalsIgnoreCase(String str)

  str -> String object being compared with the invoking string object

  returns -> true if the strings contain the same characters in the same order.

- Example

```
class EqualsDemo {
        public static void main(String args[ ]) {
                String s1 = “Hello”;
                String s2 = “Hello”;
                String s3 = “Hi”;
                String s4 = “HELLO”;
                System.out.println(s1.equals(s2);
                System.out.println(s1.equals(s3);
                System.out.println(s1.equals(s4);
                System.out.println(s1.equalsIgnoreCase(s4);
        }
}
```

❑ regionMatches( )

- compares a specific region inside a string with another specific region in another string.
- General Forms:

boolean regionMatches(int startIndex, String str2, int str2StartIndex,
int numChars)

boolean regionMatches(boolean ignoreCase, int startIndex, String str2,
int str2StartIndex, int numChars)

startIndex -> index at which the comparison will start at within str1.

str2 -> string being compared.

str2StartIndex ->index at which the comparison will start at within str2.

numChars -> length of the substring being compared.

ignoreCase -> if it is true, the case of the characters is ignored.

- Example

```
String s1 = “This is a test”;
String s2 = “This can be a TEST”;
int start = 10;
int start1 = 14;
int numChars = 4;
System.out.println(s1.regionMatches(start, s2,start1,numChars));
int pos =10;
int pos1 = 14;
System.out.println(s1.regionMatches(true, pos, s2,start1,numChars));
```

❑ startsWith( ) and endsWith( )

- startsWith( ) -> determines whether a given string begins with a specified string
- endsWith( ) -> determines whether the given string ends with a specified string
- General forms:

  boolean startsWith(String str)

  boolean endsWith(String str)

- Example

  “Foobar”.endsWith(“bar”) -> true

  “Foobar”.startsWith(“Foo”) -> true

- Another form of startsWith( )

  boolean startsWith(String str, int startIndex)

  startIndex -> index into the invoking object at which point, the search will begin

  “Foobar”.startsWith(“bar”, 3) -> true

❑ equals( )

- equals( ) method compares the characters inside a String object.
- Example

```
String s1 = “Hello”;
String s2 = new String(s1);
System.out.println(s1.equals(s2));
```

Output

true

❑ compareTo( )

- For sorting, we need to know which string is less than, equal to, or greater than the next.
- A String is less than another if it comes before the other in dictionary order.
- A String is greater than another if it comes after the other in dictionary order.
- General form

```
int compareTo(String str)
```

str -> string being compared

Value	Meaning
< 0	Invoking string less than str
> 0	Invoking string greater than str
= 0	Two strings are equal

- Example

- Now comes first because of the uppercase(uppercase has low value in ascii set)
- If you want ignore the case while comparing, then call the method,

compareToIgnoreCase( )

GF

int compareToIgnoreCase(String str)

Searching Strings

- indexOf( ) -> searches for the first occurrence of a character or substring.
- lastIndexOf( ) -> searches for the last occurrence of a character or substring
- General forms

  int indexOf(char ch)

  int lastIndexOf(char ch)

  int indexOf(String str)

  int lastIndexOf(String str)

  ch -> character being sought

  str -> substring

- Specifying starting points for the search

  int indexOf(char ch, int startIndex)

  int latIndexOf(char ch, int startIndex)

  startIndex -> index at which point the search begins

  -> for index( ) search runs from startIndex to the end of the string.

  -> for lastIndexOf( ), search runs from startIndex to zero.

- Example

```
String s = “This is a test.This is too”;
System.out.println(s.indexOf(‘t’));
System.out.println(s.lastIndexOf(‘t’));
System.out.println(s.indexOf(“is”);
System.out.println(s.indexOf(‘s’,10));
System.out.println(s.lastIndexOf(“is”, 15));
```

Modifying a String

- String objects are immutable. To modify a string,

  -> use one of the String methods given below

❖ **subString( )**

- To extract a sub string
- 2 forms

  String subString(int startIndex)

  -> from startIndex to end of the invoking string

  String subString(int startIndex, int endIndex)

  -> from startIndex to endIndex – 1

- Example

❖ concat( )

- To concatenate two strings
- General form

  String concat(String str)

- Example

```
String s1 = “One”;
String s2 = s1.concat(“Two”);
System.out.println(s2);
```

Output

One Two

❖ replace( )

- Replaces all occurrences of one character in the invoking string with another character.
- General form

  String replace(char original, char replacement)

- Example

  String s = “Hello”;

  String s1 = s.replace(‘l’, ‘w’);

  System.out.println(s1);

Output

Hewwo

❖ trim( )

- Returns a copy of the invoking string from which any leading and trailing whitespace has been removed.
- General form

  String trim( )
- Example

  String s = “ Hello World “.trim( );

  System.out.println(s);

Output

Hello World

❖ Changing the case of characters

- toLowerCase( ) -> converts all the characters in a String from uppercase to lowercase
- toUpperCase( ) -> converts all the characters in a String from lowercase to uppercase
- General forms

  String toLowerCase( )

  String toUpperCase( )

- Example

```
String s = “This is a test”;
String upper = s.toUpperCase( );
String lower = s.toLowerCase( );
System.out.println(upper);
System.out.println(lower);
```

Output

THIS IS A TEST

this is a test

Data Conversion using valueOf( )

- The valueOf( ) method converts data from its internal format into a human readable form.
- static method
- General forms

  static String valueOf(double num)

  static String valueOf(long num)

  static String valueOf(Object ob)

  static String valueOf(char chars[ ])
- valueOf( ) is called when String representation of some other data type is needed.

- Example

```
String s = “hello”;
int a = 10;
String abc = s.valueOf(a);
System.out.println(abc);
```

Output

10