Characters and Strings

- The Java platform contains three classes

1. **Character**: **-** A class whose instances can hold a single character value
2. **String**: **-** A class for working with immutable (unchanging) data composed of multiple characters
3. **StringBuffer**: **-** A class for storing and manipulating mutable data composed of multiple characters.

- **Character**
  - use a Character object instead of a primitive char variable when an object is required
  - **For example**, when passing a character value into a method

```
public class CharacterDemo {
public static void main(String args[]) {
        Character a = new Character('A');
        Character a2 = new Character('a');
        Character b = new Character('b');
        int difference = a.compareTo(b);
        if (difference == 0) {
                        System.out.println(a+ " is equal to "+ b);
        }
        else if (difference <0)
                        System.out.println(a+" is less than "+b );
        else
            System.out.println(a+" is greater than "+b);
```

```
System.out.println(a+" is " + ((a.equals(a2)) ? "equal" : "not
                                                    equal") + "
   to "+a2);
System.out.println("The character " + a.toString() + " is " +
            (Character.isUpperCase(a.charValue()) ? "upper" :
                                                    "lower")
   + "case.");
}
}
```

- **Output**
  - A is less than b
  - A is not equal to a
  - The character A is uppercase.

Character Class

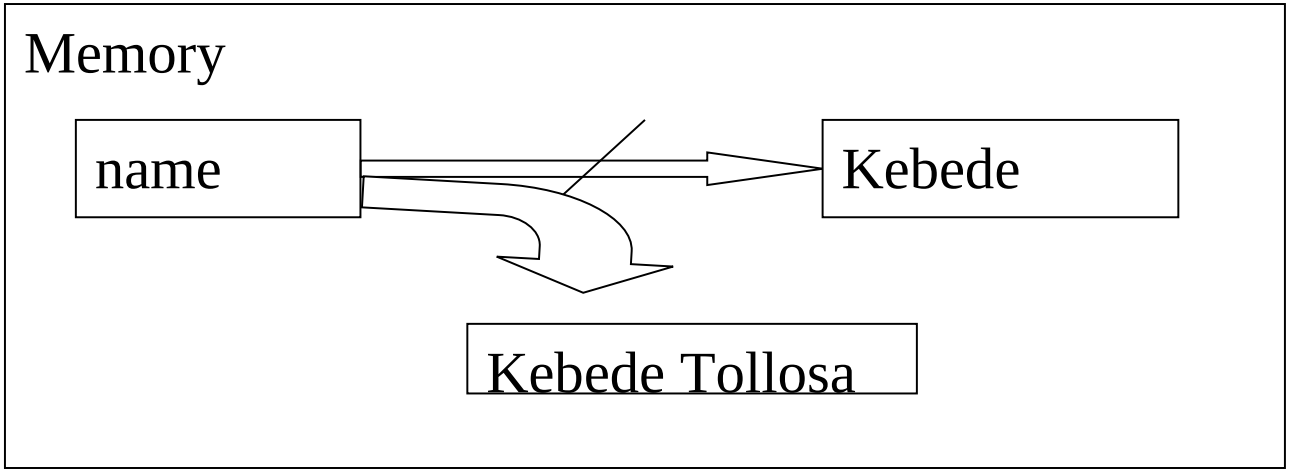
- **Constant**
  - public static final int MAX_RADIX = 36
  - public static final char MAX_VALUE =‘\ffff’
  - public static final int MIN_RADIX = 2
  - public static final char MIN_VALUE =’\0000’
- **Method**
  - **Character(char value)** : Constructor to initialize the object as value
  - **char charValue()** : Convert into char type
  - **static boolean isDigit(char ch)** : Test whether is digit?
  - **static boolean isLetter(char ch)** : Test whether is letter?
  - **static boolean isLetterOrDigit(char ch)** : Return when it is letter or digit.

- **String**
  - In Java a **string** is a sequence of characters
  - Java implements strings as objects of type **String.**
  - objects of type **String are** immutable;
  - once a **String** object is created, its contents cannot be altered
  - If you want to change **String** object
    - create a new string that contains the modifications.
    - Use StringBuffer class
  - Strings can be constructed a variety of ways.
    - use a statement like:

      String myString = "this is a test";
    - using the **new** keyword and a constructor
      - For example: String str = new String (charArray);
  - Both **String and StringBuffer** are declared **final**, which means that neither of these classes may be subclassed.

- Strings are immutable

```
class String Test{
   public static void main(String args[]){
   String name = “Kebede”;
   System.out.println(name);
   name =  name + “Tollosa”;
   System.out.println(name);
   }
}
```



- OutPut:

  Kebede

  Kebede Tollosa

The String Constructors

- The **String** class supports several constructors.

  String s = new String();

  - To create a **String** initialized by an array of characters, use the constructor

    String(char *chars[ ])

  - example:

    ```
    char chars[] = { 'a', 'b', 'c' };
    String s = new String(chars);
    ```

  - This constructor initializes **s** with the string **"abc".**
  - You can specify a sub-range of a character array as an initializer using the following constructor:

    String(char chars[ ], int startIndex, int numChars)

  - **startIndex** specifies the index at which the subrange begins, and **numChars** specifies the number of characters to use

- **example**:

```
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
String s = new String(chars, 2, 3);
```

- This initializes **s** with the characters **cde.**
- You can construct a **String** object that contains the same character sequence as another String object using this constructor:

  String(String *strObj)*

- example:

```
class MakeString {
  public static void main(String args[]) {
  char c[] = {'J', 'a', 'v', 'a'};
  String s1 = new String(c);
  String s2 = new String(s1);
  System.out.println(s1);
  System.out.println(s2);
 } }
```

- The output from this program is as follows:

  Java

  Java

- The following program converts the given **byte** array into **String**

```
// Construct string from subset of char array.
class SubStringCons {
   public static void main(String args[]) {
   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);
   }  }
```

- This program generates the following output:

  ABCDEF

  CDE

String Class

- **Method**
  - **char charAt(int index)** :
    - Return the character at specific position in string
  - **int length()** :
    - Return the length of string
  - **String toUpperCase()** :
    - Convert the string into upper character
  - **String toLowerCase()** :
    - Convert the string into lower character
  - **void getChars()** :
    - extract more than one character at a time
  - **String substring(int beginIndex)** :
    - Make the sub string from beginIndex to the end of the string

- **getChars( )**
  - extract more than one character at a time
  - general form:

    void getChars(int *Start*, *int End*, *char target[ ]*, *int targetStart)*
  - ***Start*** specifies the index of the beginning of the substring,
  - ***End*** specifies an index that is one past the end of the desired substring
  - The array that will receive the characters is specified by ***target***.
  - The index within target at which the substring will be copied is passed in ***targetStart***

- The following program demonstrates **getChars( ):**

```
class getCharsDemo {
    public static void main(String args[]) {
                        String s = "This is a demo of the getChars
    method.";
                        int start = 10;
                        int end = 14;
                        char buf[] = new char[end - start];
                        s.getChars(start, end, buf, 0);
                        System.out.println(buf);
    }
}
```

- Here is the output of this program:

```
demo
```

- **String Concatenation**
  - **+ operator**
  - Example:

```
String age = "9";
String s = "He is " + age + " years old.";
System.out.println(s);
```

- **String Concatenation with Other Data Types**

```
int age = 9;
String s = "He is " + age + " years old.";
System.out.println(s);
```

  - The compiler will convert an operand to its string equivalent whenever the other operand of the **+** is an instance of String.

- Consider the following:

```
String s = "four: " + 2 + 2;
System.out.println(s);
```

- This fragment displays

  four: 22

- To complete the integer addition first, you must use parentheses, like this:

  String s = "four: " + (2 + 2);

- Now **s** contains the string **"four: 4".**

- You can concatenate two strings using **concat( )**

  String concat(String *str)*

```
String s1 = "one";
String s2 = s1.concat("two");
```

- **String Conversion and toString( )**
  - Java converts data into its string representation during concatenation by
    - calling one of the string conversion method **valueOf( )**
    - **valueOf( )** is overloaded for all the simple types and for type Object
  - For the simple types, **valueOf( )** returns a string that contains the human-readable equivalent of the value with which it is called.
  - For objects, **valueOf( )** calls the **toString( )** method on the object
  - Every class implements **toString( )** because it is defined by **Object**
  - The **toString( )** method has this general form**:**

String toString( )

- The following program demonstrates overriding **toString( )** for the Box class:

```
// Override toString() for Box class.
class Box {
   double width, height, depth;
   Box(double w, double h, double d) {
   width = w;
   height = h;
   depth = d;
    }
   public String toString() {
   return "Dimensions are " + width + " by " + depth + " by "
   + height + ".";
   }
   }
```

```
class toStringDemo {
    public static void main(String args[]) {
                    Box b = new Box (10, 12, 14);
                    String s = "Box b: " + b; // concatenate Box
  object
                    System.out.println (b); // convert Box to string
                    System.out.println (s);
    }
}
```

- The output of this program is shown here:

  Dimensions are 10 by 14 by 12.

  Box b: Dimensions are 10 by 14 by 12

```
// A bubble sort for Strings.
class SortString {
static String arr[] = {"Now", "is", "the", "time", "for", "all", "good",
   “persons", "to", "come", "to", "the", "aid", "of", "their",
   "country”};
public static void main(String args[]) {
   for(int j = 0; j < arr.length; j++) {
   for(int i = j + 1; i < arr.length; i++) {
   if(arr[i].compareTo(arr[j]) < 0) {
   String t = arr[j];
   arr[j] = arr[i];
   arr[i] = t;
   }
   }
System.out.println(arr[j]);
}
} }
```

- **Searching Strings**
  - The **String** class provides two methods that allow you to search a string for a specified character or substring:
  - **indexOf( )** Searches for the first occurrence of a character or substring.
  - **lastIndexOf( )** Searches for the last occurrence of a character or substring.
  - To search for the first or last occurrence of a substring, use

    int indexOf(String *str)*

    int lastIndexOf(String *str)*
  - **str** specifies the substring
  - You can specify a starting point for the search using these forms:

    int indexOf(chae *ch, int startIndex)*

    int lastIndexOf(char *ch, int startIndex)*

    int indexOf(String *str, int startIndex)*

    int lastIndexOf(String *str, int startIndex)*

- For **indexOf( ),** the search runs from *startIndex* to the end of the string.
- *For **lastIndexOf( ),*** the search runs from startIndex to zero.

```
// Demonstrate indexOf() and lastIndexOf().
class indexOfDemo {
public static void main(String args[]) {
  String s = "Now is the time for all good persons" + "to come to the aid of their country.";
  System.out.println(s);
  System.out.println("indexOf(t) = " + s.indexOf('t'));
  System.out.println("lastIndexOf(t) = " + s.lastIndexOf('t'));
  System.out.println("indexOf(the) = " + s.indexOf("the"));
  System.out.println("lastIndexOf(the) = " + s.lastIndexOf("the"));
```

```
System.out.println("indexOf(t, 10) = " + s.indexOf('t', 10));
System.out.println("lastIndexOf(t, 60) = " + s.lastIndexOf('t', 60));
System.out.println("indexOf(the, 10) = " + s.indexOf("the", 10));
System.out.println("lastIndexOf(the, 60)=“+s.lastIndexOf("the",60));
}
}
```

- the output of this program:

  Now is the time for all good men to come to the aid of their country.

- Now is the time for all good personsto come to the aid of their country.

  indexOf(t) = 7 lastIndexOf(t) = 68

  indexOf(the) = 7 lastIndexOf(the) = 58

  indexOf(t, 10) = 11 lastIndexOf(t, 60) = 58

  indexOf(the, 10) = 47 lastIndexOf(the, 60)=58

- **Modifying a String**
  - to modify a **String,** you must either copy it into a **StringBuffer or** use one of the following String methods**,**
  - **substring( )**
    - It has two forms.
      1. String substring(int *startIndex)*
      2. String substring(int *startIndex, int endIndex)*
    - The string returned contains all the characters from the beginning index, up to, but not including, the ending index.

```
// Substring replacement.
class StringReplace {
   public static void main(String args[]) {
                        String org = "This is a test. This is, too.";
                        String search = "is";
                        String sub = "was";
                        String result = "";
                        int i;
                        do { // replace all matching substrings
                                System.out.println(org);
                                i = org.indexOf(search);
                                if(i != -1) {
                                        result = org.substring(0, i);
                                        result = result + sub;
```

```
                    result = result + org.substring(i +
  search.length());
                    org = result;
                }//end of if
  } while(i != -1);
}
}
```

- The output

```
This is a test. This is, too.
Thwas is a test. This is, too.
Thwas was a test. This is, too.
Thwas was a test. Thwas is, too.
Thwas was a test. Thwas was, too.
```

Exercise

- Write a program that performs the following
  - Accepts a string
  - converts it into Uppercase or Lowercase
  - extract substring based on the users interest.
  - Search a Palindrome word within the given string.
  - counts the number of words
  - Extract the special characters

StringBuffer Class

- is a peer class of String
- represents growable and writeable character sequences.
- will automatically grow to make room for such additions
- often has more characters pre-allocated than are actually needed, to allow room for growth.
  - for example, when reading text data from a file.
- may have characters and substrings inserted in the middle or appended to the end.
- **StringBuffer** defines these three constructors:
  - StringBuffer( )
  - StringBuffer(int *size)*
  - StringBuffer(String *str)*

- The default constructor (the one with no parameters) reserves room for **16** characters without reallocation.
- The second version accepts an integer argument that explicitly sets the size of the buffer.
- The third version accepts a **String** argument that sets the initial contents of the **StringBuffer** object and reserves room for **16** more characters without reallocation.
- By allocating room for a few extra characters, **StringBuffer** reduces the number of reallocations that take place.

- **length( ) and capacity( )**
  - The current length of a **StringBuffer** can be found via the **length( )** method
  - The total allocated capacity can be found through the **capacity( ) method.**
- Here is an example:

```
// StringBuffer length
class StringBufferDemo1 {
   public static void main(String args[]) {
                        StringBuffer sb = new StringBuffer("Hello");
                        System.out.println("buffer = " + sb);
                        System.out.println("length = " + sb.length());
                         System.out.println("Capacity= "+ sb.capacity());
   } }
```

- **Output**: buffer = Hello

  length = 5 Capacity = 21

- **append( )**
  - The **append()** method concatenates the string representation of any other type of data to the end of the invoking **StringBuffer** object
  - few of its forms:

    StringBuffer append(String *str)*

    StringBuffer append(int *num)*

    StringBuffer append(Object *obj)*

```
// Demonstrate append().
class appendDemo {
    public static void main(String args[]) {
            String s;
            int a = 42;
            StringBuffer sb = new StringBuffer(40);
            s = sb.append("a = ").append(a).append("!").toString();
            System.out.println(s);
    }
}
```

- The output of this example is shown here:

  a = 42!

- **insert( )**
  - The **insert( )** method inserts one string into another
  - few of its forms:

```
StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
StringBuffer insert(int index, Object obj)
// Demonstrate insert().
class insertDemo {
   public static void main(String args[]) {
   StringBuffer sb = new StringBuffer("I Java!");
   sb.insert(2, "like ");
   System.out.println(sb);
   }
}
```

  - The output

```
I like Java!
```

- **reverse( )**

  StringBuffer reverse( )

  // Using reverse() to reverse a StringBuffer.

  class ReverseDemo {

  public static void main(String args[]) {

  StringBuffer s = new StringBuffer("abcdef");

  System.out.println(s);

  s.reverse();

  System.out.println(s);

  }

  }

  – the output:

  abcdef

  fedcba

- **delete( ) and deleteCharAt( )**
  - The **delete( )** method deletes a sequence of characters from the invoking object**.**

    StringBuffer delete(int *startIndex*, *int endIndex)*
  - *startIndex* specifies the index of the first character to remove, and *endIndex* specifies an index one past the last character to remove.
  - the substring deleted runs from ***startIndex to endIndex–1***.

    StringBuffer deleteCharAt(int *loc)*
  - deletes the character at the index specified by *loc*.

```
// Demonstrate delete() and deleteCharAt()
class deleteDemo {
  public static void main(String args[]) {
                    StringBuffer sb = new StringBuffer("This is a
  test.");
                    sb.delete(4, 7);
                    System.out.println("After delete: " + sb);
                    sb.deleteCharAt(0);
                    System.out.println("After deleteCharAt: " + sb);
  }
}
```

- The output:

```
After delete: This a test.
After deleteCharAt: his a test.
```

- **replace( )**
  - replaces one set of characters with another set inside a **StringBuffer** object**.**

    StringBuffer replace(int *startIndex, int endIndex, String str)*

```
// Demonstrate replace()
class replaceDemo {
   public static void main(String args[]) {
   StringBuffer sb = new StringBuffer("This is a test.");
   sb.replace(5, 7, "was");
   System.out.println("After replace: " + sb);
   }
}
```

  - The output:

    After replace: This was a test.

StringBuffer Class

```
String now = new java.util.Date().toString();
StringBuffer strbuf = new StringBuffer(now);
strbuf.append(" : ").append(now.length()).append('.');
System.out.println(strbuf.toString());
```