Unit V

Strings, Streams and Files

String Class

- *String* is a sequence of characters.
- In Java string is not an array but, string is an objects of type **String**.
- String object is created using String class (java.lang package).
- The java.lang.String class is a final class which handles string as a set of Unicode characters.

- Strings are immutable in Java.
- Means the contents of a String object can not be modified.
- Any change made to a String object will result a new object, keeping the old one unchanged.

The String Constructors

- To create an empty **String**

```
String s = new String();
```

- To create a **String** initialized by an array of characters

```
String(char chars[ ])
```

- Initializing with a range of a character array

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

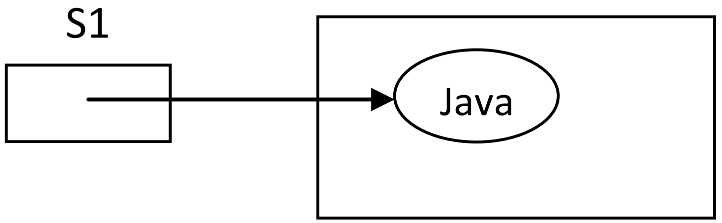
```
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };

String s = new String(chars, 2, 3);
```

- String can also be created using string literal

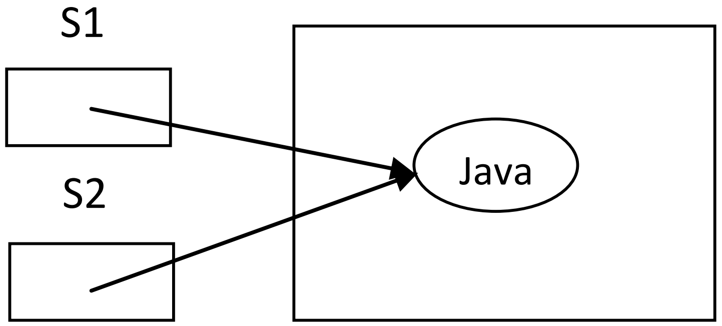
```
String s2 = "abc";
```

Statement | Effect

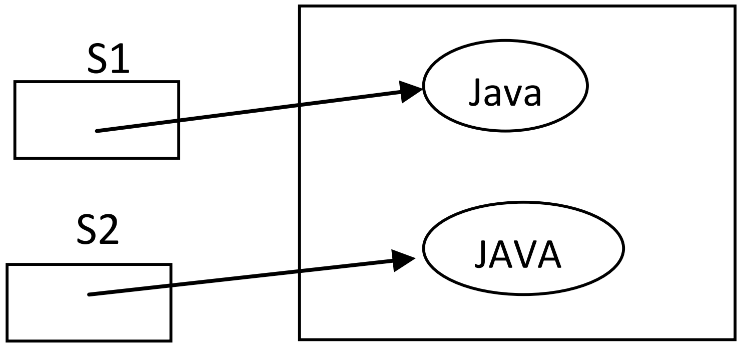
```
String s1 = new
        String("Java");
```



```
String s2 = s1;
```



```
S2 = s1.toUpperCase();
```



The String Operations

length( ) - The length of a string is the number of characters that it contains.

```
char chars[] = { 'a', 'b', 'c' };

String s = new String(chars);

System.out.println(s.length()); // 3
```

charAt( ) - extract a single character from a String. This function accept index as a parameter.

```
char charAt(int where)
```

```
char ch;

String s = “abc”;

ch = s.charAt(1); // assigns the value “b” to ch.
```

equals( ) - To compare two strings for equality

```
boolean equals(Object str)

String s1 = "Hello";

String s2 = "Hello";

System.out.println(s1.equals(s2)); // true
```

equals( ) Versus ==

- the **equals( )** method compares the characters inside a **String** object.
- The **==** operator compares two object references to see whether they refer to the same instance.

startsWith( ) and endsWith( ) -

```
boolean startsWith(String str)
boolean endsWith(String str)
```

```
String s = “Computer”;
s.endsWith("ter")
```

and

```
s.startsWith("Com")
```

are both **true**.

indexOf( ) Searches for the first occurrence of a character or substring.

```
int indexOf(String str)

String s = “I am a programmer”;

System.out.println(s.indexOf('am')); // 2
```

lastIndexOf( ) Searches for the last occurrence of a character or substring.

```
int lastIndexOf(String str)

String s = “I am a programmer”;

System.out.println(s. lastIndexOf ('am')); // 12
```

substring( ) - This method is used to extract a substring from a string.

```
String substring(int startIndex, int endIndex)

String s = “abcdef”;

System.out.println(s. substring(2,4)); // cd
```

concat( ) – This method is used to concatenate two strings.

```
String concat(String str)

String s1 = "one";

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

replace( ) – This method is used to replace one character with another.

```
String replace(char original, char replacement)

String s = "Hello".replace('l', 'w'); // Hewwo
```

trim( ) – This method returns a copy of the invoking string from which any leading and trailing whitespace has been removed. It has this general form:

```
String trim( )
```

Here is an example:

```
String s = "   Hello World   ".trim();
```

toLowerCase( ) - converts all the characters in a string from uppercase to lowercase

```
String toLowerCase( )

String s = "Java";

String upper = s.toUpperCase();

System.out.println("Uppercase: " + upper); // JAVA
```

toUpperCase( ) - method converts all the characters in a string from lowercase to uppercase.

```
String toUpperCase( )

String s = "Java";

String lower = s. toLowerCase();

System.out.println("Lowercase: " + lower); // java
```

StringBuffer Class

- The StringBuffer class is used to create string objects which can be modified.
- Defined in the java.lang package
- It is a final class.
- It represents a dynamic string.
- Java allocates extra memory for the StringBuffer object to allow growth.

StringBuffer Constructors

- It creates an empty object having initial capacity 16

```
StringBuffer( )
```

- It creates object of a specific size

```
StringBuffer(int size)
```

- It creates object using String object, additional capacity of 16 chars

```
StringBuffer(String str)
```

The StringBuffer Operations

- `int length( )`
  - find current length of a **StringBuffer** object
- `int capacity( )`
  - find total allocated capacity

append( )

The **append( )** method concatenates the string representation of any other type of data to the end of the invoking **StringBuffer** object.

```
StringBuffer append(String str)

StringBuffer append(int num)

StringBuffer append(Object obj)
```

insert( )

The **insert( )** method inserts one string into another.

```
StringBuffer insert(int index, String str)

StringBuffer insert(int index, char ch)

StringBuffer insert(int index, Object obj)
```

Here, *index* specifies the index at which point the string will be inserted into the invoking **StringBuffer** object.

reverse( )

You can reverse the characters within a **StringBuffer** object

```
      StringBuffer reverse( )
```

delete( )

method deletes a sequence of characters from the invoking object.

```
      StringBuffer delete(int startIndex, int endIndex)
```

the substring deleted runs from *startIndex* to *endIndex*–1.

Introduction to java.io package

- Support for I/O operations
- Java I/O classes
  - Network connection
  - Memory buffer
  - Disk file
- These devices are all handled by the *stream.*

classes defined by **java.io**

BufferedInputStream, FileWriter, PipedOutputStream, BufferedOutputStream, FilterInputStream, PipedReader,

BufferedReader, FilterOutputStream, PipedWriter, BufferedWriter, FilterReader, PrintStream,

ByteArrayInputStream, FilterWriter, PrintWriter, ByteArrayOutputStream, InputStream, PushbackInputStream,

CharArrayReader, InputStreamReader, PushbackReader, CharArrayWriter, LineNumberReader, RandomAccessFile

Interfaces are defined by **java.io**

Closeable, FileFilter, ObjectInputValidation, DataInput, FilenameFilter, ObjectOutput,

DataOutput, Flushable, ObjectStreamConstants, Externalizable, ObjectInput, Serializable

File

- **File** class does not operate on streams.
- It describes the properties of a file itself.
- A **File** object is used to obtain or manipulate the inform
  - Permissions
  - Time
  - Date
  - Directory path, and
  - Navigate subdirectory hierarchies

File constructors

```
File(String directoryPath)

File(String directoryPath, String
  filename)

File(File dirObj, String filename)

File(URI uriObj)
```

Directories

- A directory is a **File** that contains a list of other files and directories.

```
String[ ] list( )
```

- A stream is a method to sequentially access a file.
- Stream essentially refers to an abstraction that is used to produce and consume flow of sequential information.
- **Stream** – A sequence of data.
  Input Stream: reads data from source.
  Output Stream: writes data to destination.

- **Byte Stream**
  - Process data byte by byte (8 bits).
  - Suitable for processing raw data like binary files.
  - For example, the *FileOutputStream* is meant for reading a raw stream of bytes, such as image data.
- **Character Stream**
  - In Java, characters are stored using Unicode conventions.
  - Character stream automatically allows us to read/write data character by character. (16 bits)
  - Character stream is useful when we want to process text files.

The Stream Classes

- Four abstract classes: **InputStream**, **OutputStream**, **Reader**, and **Writer**.
- The programs perform their I/O operations through concrete subclasses.
- **InputStream** and **OutputStream** are designed for byte streams.
- **Reader** and **Writer** are designed for character streams.

The Byte Streams

- **InputStream**
  - An abstract class that defines byte input stream.
  - Implements the **Closeable** interface.
  - Most of the methods in this class will throw an **IOException** on error conditions.
- **OutputStream**
  - **OutputStream** is an abstract class that defines streaming byte output.
  - It implements the **Closeable** and **Flushable** interfaces.
  - Most of the methods in this class return **void** and throw an **IOException** in the case of errors.

The Methods Defined by InputStream

Methods	Descreption
int available( )	Returns the number of bytes of input currently available for reading.
void close( )	Closes the input source. Further read attempts will generate an IOException.
int read( )	Returns an integer representation of the next available byte of input. –1 at eof.
int read(byte buffer[ ])	Attempts to read up to buffer.length bytes into buffer and returns the actual number of bytes that were successfully read. –1 at eof.
int read(byte buffer[ ], int offset, int numBytes)	Attempts to read up to numBytes bytes into buffer starting at buffer[offset], returning the number of bytes successfully read. –1 at eof.

The Methods Defined by OutputStream

Methods	Descreption
void close( )	Closes the output stream. Further write attempts will generate an IOException.
void flush( )	Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.
void write(int b)	Writes a single byte to an output stream.
void write(byte buffer[ ])	Writes a complete array of bytes to an output stream.
void write(byte buffer[ ], int offset,int numBytes)	Writes a subrange of numBytes bytes from the array buffer, beginning at buffer[offset].

FileInputStream

- The **FileInputStream** class creates an **InputStream** that you can use to read bytes from a file.
- Its two most common constructors are shown here:

```
FileInputStream(String filepath)

FileInputStream(File fileObj)
```

- Either can throw a **FileNotFoundException**.

FileOutputStream

- **FileOutputStream** creates an **OutputStream** that you can use to write bytes to a file. Its most commonly used constructors are shown here:

```
FileOutputStream(String filePath)
FileOutputStream(File fileObj)
FileOutputStream(String filePath,
  boolean append)
FileOutputStream(File fileObj, boolean
  append)
```

- They can throw a **FileNotFoundException**.

ByteArrayInputStream

- An input stream
- Uses a byte array as the source.

ByteArrayOutputStream

- An output stream
- Uses a byte array as the destination

BufferedInputStream

- Attach a memory buffer to the input streams.
- To improve performance

BufferedOutputStream

- Attach a memory buffer to the output streams.
- To improve performance
- Add flush() method

PrintStream

- The PrintStream class provides methods to write data to another stream.
- The PrintStream class automatically flushes the data so there is no need to call flush() method.
- Its methods don't throw IOException
- It implements the Appendable, Closeable, and Flushable interfaces.

DataOutputStream and DataInputStream

- Write or read primitive data to or from a stream
- Implement the DataOutput and DataInput interfaces, respectively
- DataOutputStream extends FilterOutputStream, which extends OutputStream

- DataOutput defines methods that convert values of a primitive type into a byte sequence and then writes it to the underlying stream.
- DataInput interface that defines methods that read a sequence of bytes and convert them into values of a primitive type.

The Character Streams

- Byte stream classes cannot work directly with Unicode characters.
- Character Stream classes directly I/O support for characters
- Top of the character stream hierarchies are the **Reader** and **Writer** abstract classes.

The Methods Defined by Reader

Methods	Descreption
abstract void close( )	abstract void close( ) Closes the input source. Further read attempts will generate an IOException.
int read( )	Returns an integer representation of the next available character from the invoking input stream. –1 is returned when the end of the file is encountered.
int read(char buffer[ ])	Attempts to read up to buffer.length characters into buffer and returns the actual number of characters that were successfully read. –1 is returned when the end of the file is encountered.
abstract int read(char buffer[ ], int offset, int numChars)	Attempts to read up to numChars characters into buffer starting at buffer[offset], returning the number of characters successfully read. –1 is returned when the end of the file is encountered.

The Methods Defined by Writer

Methods	Descreption
Writer append(char ch)	Appends ch to the end of the invoking output stream. Returns a reference to the invoking stream.
Writer append(CharSequence chars)	Appends chars to the end of the invoking output stream. Returns a reference to the invoking stream.
Writer append(CharSequence chars, int begin, int end)	Appends the subrange of chars specified by begin and end–1 to the end of the invoking ouput stream. Returns a reference to the invoking stream.
abstract void close( )	Closes the output stream. Further write attempts will generate an IOException.
abstract void flush( )	Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.

The Methods Defined by Writer

Methods	Descreption
void write(int ch)	Writes a single character to the invoking output stream. Note that the parameter is an int, which allows you to call write with expressions without having to cast them back to char.
void write(char buffer[ ])	Writes a complete array of characters to the invoking output stream.
abstract void write(char buffer[ ], int offset, int numChars)	Writes a subrange of numChars characters from the array buffer, beginning at buffer[offset] to the invoking output stream.
void write(String str)	Writes str to the invoking output stream.
void write(String str, int offset, int numChars)	Writes a subrange of numChars characters from the string str, beginning at the specified offset.

FileReader

- Use to read the contents of a file
- Two most commonly used constructors are shown here:

```
FileReader(String filePath)
FileReader(File fileObj)
```

- Either can throw a FileNotFoundException.
- Here, filePath is the full path name of a file, and fileObj is a File object that describes the file.

FileWriter

- Use to write to a file
- Most commonly used constructors are shown here:

```
FileWriter(String filePath)
FileWriter(String filePath, boolean
append)
FileWriter(File fileObj)
FileWriter(File fileObj, boolean
append)
```

- They can throw an IOException.
- Here, filePath is the full path name of a file, and fileObj is a File object that describes the file.
- If append is true, then output is appended to the end of the file.

CharArrayReader

- An input stream that uses a character array as the source.

  ```
  CharArrayReader(char array[ ])
  CharArrayReader(char array[ ], int
  start, int numChars)
  ```

- Here, array is the input source.
- The second constructor creates a Reader from a subset of your character array that begins with the character at the index specified by start and is numChars long.

RandomAccessFile

- It is used to access file randomly.
- It is not derived from InputStream or OutputStream. Instead,
- it implements the interfaces DataInput and DataOutput, which define the basic I/O methods.
- It also implements the Closeable interface
- It can position the file pointer within the file.

- It has these two constructors:

```
RandomAccessFile(File fileObj,
String access)
throws FileNotFoundException

RandomAccessFile(String filename,
String access)
throws FileNotFoundException
```

- `access` determines what type of file access is permitted

access	**Meaning**
“r”	The file can be read, but not written
“rw”	The file is opened in read-write mode
“rws”	The file is opened for read-write operations and every change to the file’s data or metadata will be immediately written to the physical device
“rwd”	The file is opened for read-write operations and every change to the file’s data will be immediately written to the physical device

- seek( )
  - is used to set the current position of the file pointer within the file

```
void seek(long newPos) throws
IOException
```

  - `newPos` specifies the new position, in bytes, of the file pointer from the beginning of the file.
  - After a call to seek( ), the next read or write operation will occur at the new file position.