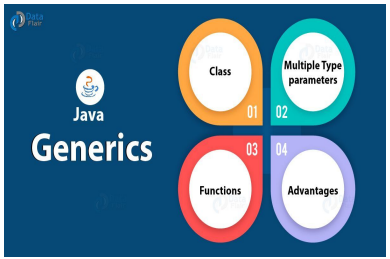
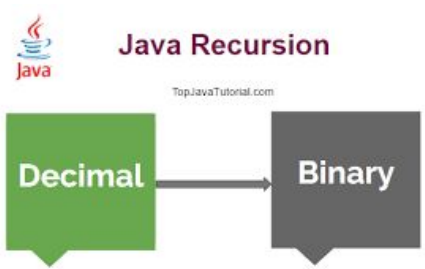
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

CHAPTER 9
BINARY I/O,RECURSION AND GENERICS





SUBJECT:OOP-I **CODE:3140705**	**PREPARED BY:** **ASST.PROF.NENSI KANSAGARA** **(CSE DEPARTMENT,ACET)**	AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

BINARY I/O

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

INTRODUCTION TO I/O PROGRAMMING

- **Java I/O** (Input and Output) is used *to process the input* and *produce the output*.
- Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
- We can perform **file handling in Java** by Java I/O API
- In this Java File IO tutorial, we show you how to read and write binary files using both legacy File I/O API and new File I/O API (NIO). The legacy API (classes in the **java.io.*** package) is perfect for manipulating low-level binary I/O operations such as reading and writing exactly one byte at a time, whereas the NIO API (classes in the **java.nio.*** package) is more convenient for reading and writing the whole file at once, and of course, faster than the old File I/O API.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

HANDLING IN JAVA

syntax:

PrintWriter o =new printwriter(“input.txt”);

o.print(“I love my country”);

Example:

Scanner i = new Scanner(new File(input.txt));

System.out.println(i.nextline());

CONCEPT OF STREAM

- A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.
- In Java, 3 streams are created for us automatically. All these streams are attached with the console.
  - **1) System.out:** standard output stream
  - **2) System.in:** standard input stream
  - **3) System.err:** standard error stream
- Let's see the code to print **output and an error** message to the console.
  - System.out.println("simple message");
  - System.err.println("error message");
- Let's see the code to get **input** from console.

```
int i=System.in.read();//returns ASCII code of 1st character

System.out.println((char)i);//will print the character
```

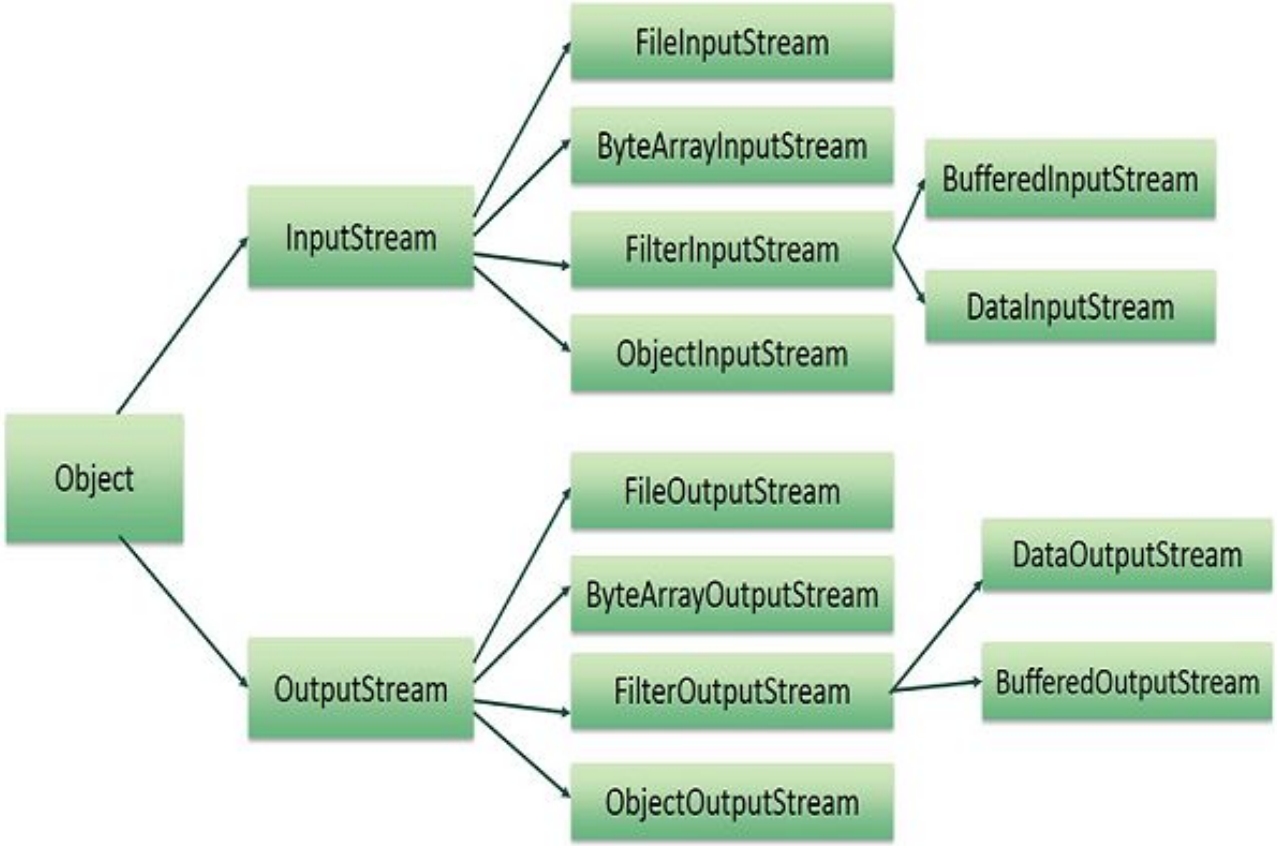
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

- The **java.io** package contains all the classes required for input output operations.
- All streams represent an input source and an output destination.
- The stream in the java.io package **supports** all the datatype **including primitive**.
- A stream can be defined as a sequence of data.
- There are two kinds of Streams
  - **InputStream** : The InputStream is used to read data from a source.
  - **OutputStream** : The OutputStream is used for writing data to a destination.

TEXT AND BINARY I/O

- All the programming languages provide support for standard I/O where the user's program can take input from a keyboard and then produce an output on the computer screen. If you are aware of C or C++ programming languages, then you must be aware of three standard devices STDIN, STDOUT and STDERR. Similarly, Java provides the following three standard streams −
  - Standard Input − This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as System.in.
  - Standard Output − This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as System.out.
  - Standard Error − This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as System.err.

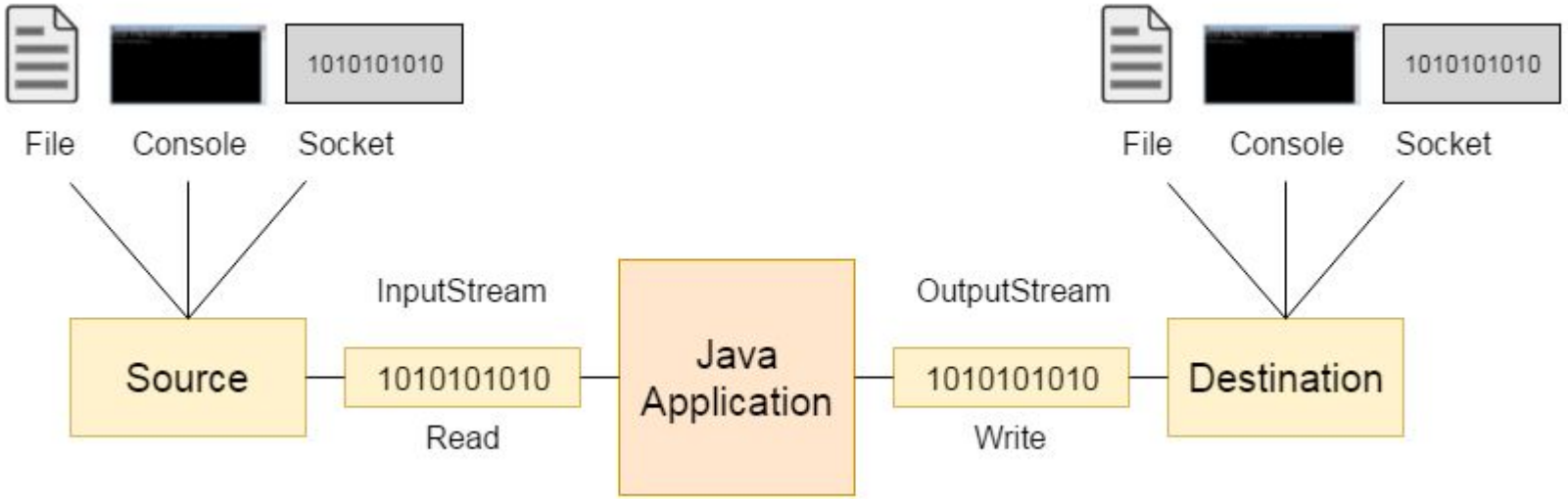
BINARY I/O CLASSES



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

FILEINPUTSTREAM

- Java FileInputStream class obtains input bytes from a file.
- It is used for reading streams of raw bytes such as image data.
- For reading streams of characters, consider using FileReader.
- It should be used to read byte-oriented data for example to read image, audio, video etc.

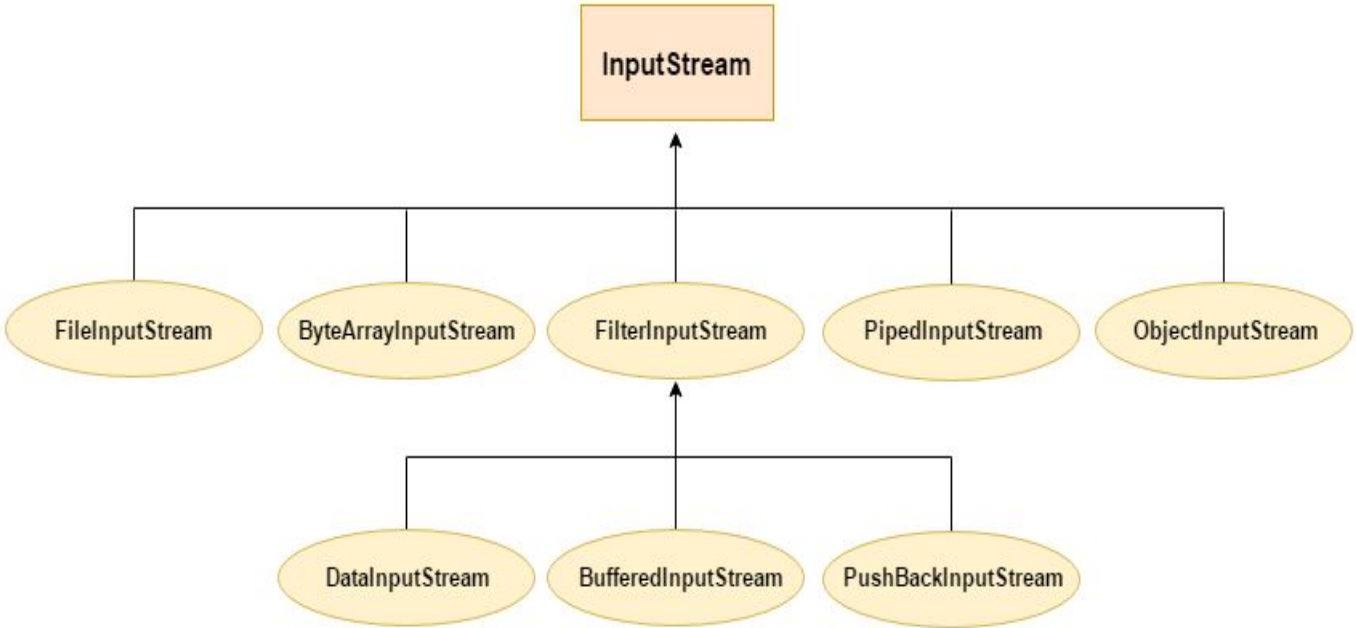


AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Java FileInputStream class declaration

Let's see the declaration for java.io.FileInputStream class:

```
1.  public class FileInputStream extends InputStream
```



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Method	Description
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream.

```
package com.javatpoint;

import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=0;
          while((i=fin.read())!=-1){
           System.out.print((char)i);
          }
          fin.close();
         }catch(Exception e){System.out.println(e);}
        }
       }
```

FILEOUTPUTSTREAM

- Java FileOutputStream is an output stream for writing data to a file.
- If you have to write primitive values then use FileOutputStream But for character-oriented data, prefer FileWriter.
- But you can write byte-oriented as well as character-oriented data.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Methods of fileoutputstream

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

```java
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
        try{
          FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
          fout.write(65);
          fout.close();
          System.out.println("success...");
         }catch(Exception e){System.out.println(e);}
    }
}
```

Output:

```
Success...
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

FILTERINPUTSTREAM

- Java FilterInputStream class implements the InputStream. It contains different sub classes as BufferedInputStream, DataInputStream for providing additional functionality. So it is less used individually.
- Java FilterInputStream class declaration
- Let's see the declaration for java.io.FilterInputStream class
  - **public class** FilterInputStream **extends** InputStream

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Java FilterInputStream class Methods

Method	Description
int available()	It is used to return an estimate number of bytes that can be read from the input stream.
int read()	It is used to read the next byte of data from the input stream.
int read(byte[] b)	It is used to read up to byte.length bytes of data from the input stream.
long skip(long n)	It is used to skip over and discards n bytes of data from the input stream.
boolean markSupported()	It is used to test if the input stream support mark and reset method.
void mark(int readlimit)	It is used to mark the current position in the input stream.
void reset()	It is used to reset the input stream.
void close()	It is used to close the input stream.

```java
import java.io.*;
public class FilterExample {
   public static void main(String[] args) throws IOException {
      File data = new File("D:\\testout.txt");
      FileInputStream  file = new FileInputStream(data);
      FilterInputStream filter = new BufferedInputStream(file);
      int k =0;
      while((k=filter.read())!=-1){
         System.out.print((char)k);
      }
      file.close();
      filter.close();
   }
}
```

FILTEROUTPUTSTREAM

- Java FileOutputStream is an output stream used for writing data to a file.
- If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.
- FileOutputStream class declaration
  - Let's see the declaration for Java.io.FileOutputStream class:
  - **public class** FileOutputStream **extends** OutputStream

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

```java
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
        try{
         FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
         fout.write(65);
         fout.close();
         System.out.println("success...");
        }catch(Exception e){System.out.println(e);}
    }
}
```

Output:

```
Success...
```

DATAINPUTSTREAM

Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.

Java application generally uses the data output stream to write data that can later be read by a data input stream.

Java DataInputStream class declaration

Let's see the declaration for java.io.DataInputStream class:

public class DataInputStream **extends** FilterInputStream **implements** DataInput

Java DataInputStream class Methods

Method	Description
int read(byte[] b)	It is used to read the number of bytes from the input stream.
int read(byte[] b, int off, int len)	It is used to read **len** bytes of data from the input stream.
int readInt()	It is used to read input bytes and return an int value.
byte readByte()	It is used to read and return the one input byte.
char readChar()	It is used to read two input bytes and returns a char value.
double readDouble()	It is used to read eight input bytes and returns a double value.
boolean readBoolean()	It is used to read one input byte and return true if byte is non zero, false if byte is zero.
int skipBytes(int x)	It is used to skip over x bytes of data from the input stream.
String readUTF()	It is used to read a string that has been encoded using the UTF-8 format.
void readFully(byte[] b)	It is used to read bytes from the input stream and store them into the buffer array.
void readFully(byte[] b, int off, int len)	It is used to read **len** bytes from the input stream.

```java
package com.javatpoint;
import java.io.*;
public class DataStreamExample {
  public static void main(String[] args) throws IOException {
    InputStream input = new FileInputStream("D:\\testout.txt");
    DataInputStream inst = new DataInputStream(input);
    int count = input.available();
    byte[] ary = new byte[count];
    inst.read(ary);
    for (byte bt : ary) {
      char k = (char) bt;
      System.out.print(k+"-");
    }
  }
}
```

Here, we are assuming that you have following data in **"testout.txt"** file:

```
JAVA
```

Output:

```
J-A-V-A
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

DATAOUTPUTSTREAM

- Java DataOutputStream class allows an application to write primitive Java data types to the output stream in a machine-independent way.
- Java application generally uses the data output stream to write data that can later be read by a data input stream.

Java DataOutputStream class declaration

- Let's see the declaration for java.io.DataOutputStream class:
- **public class** DataOutputStream **extends** FilterOutputStream **implements** DataOutput

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Method	Description
int size()	It is used to return the number of bytes written to the data output stream.
void write(int b)	It is used to write the specified byte to the underlying output stream.
void write(byte[] b, int off, int len)	It is used to write len bytes of data to the output stream.
void writeBoolean(boolean v)	It is used to write Boolean to the output stream as a 1-byte value.
void writeChar(int v)	It is used to write char to the output stream as a 2-byte value.
void writeChars(String s)	It is used to write string to the output stream as a sequence of characters.
void writeByte(int v)	It is used to write a byte to the output stream as a 1-byte value.
void writeBytes(String s)	It is used to write string to the output stream as a sequence of bytes.
void writeInt(int v)	It is used to write an int to the output stream
void writeShort(int v)	It is used to write a short to the output stream.
void writeShort(int v)	It is used to write a short to the output stream.
void writeLong(long v)	It is used to write a long to the output stream.
void writeUTF(String str)	It is used to write a string to the output stream using UTF-8 encoding in portable manner.
void flush()	It is used to flushes the data output stream.

```java
package com.javatpoint;

import java.io.*;
public class OutputExample {
    public static void main(String[] args) throws IOException {
        FileOutputStream file = new FileOutputStream(D:\\testout.txt);
        DataOutputStream data = new DataOutputStream(file);
        data.writeInt(65);
        data.flush();
        data.close();
        System.out.println("Succcess...");
    }
}
```

Output:

```
Succcess...
```

testout.txt:

```
A
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

BUFFEREDINPUTSTREAM

- Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.
- The important points about BufferedInputStream are:
  - When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.
  - When a BufferedInputStream is created, an internal buffer array is created.

Java BufferedInputStream class declaration

- Let's see the declaration for Java.io.BufferedInputStream class:
- **public class** BufferedInputStream **extends** FilterInputStream

Java BufferedInputStream class methods

Method	Description
int available()	It returns an estimate number of bytes that can be read from the input stream without blocking by the next invocation method for the input stream.
int read()	It read the next byte of data from the input stream.
int read(byte[] b, int off, int ln)	It read the bytes from the specified byte-input stream into a specified byte array, starting with the given offset.
void close()	It closes the input stream and releases any of the system resources associated with the stream.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readlimit)	It sees the general contract of the mark method for the input stream.
long skip(long x)	It skips over and discards x bytes of data from the input stream.
boolean markSupported()	It tests for the input stream to support the mark and reset methods.

```
import java.io.*;
public class BufferedInputStreamExample{
 public static void main(String args[]){
  try{
    FileInputStream fin=new FileInputStream("D:\\testout.txt");
    BufferedInputStream bin=new BufferedInputStream(fin);
    int i;
    while((i=bin.read())!=-1){
     System.out.print((char)i);
    }
    bin.close();
    fin.close();
  }catch(Exception e){System.out.println(e);}
 }
}
```

Here, we are assuming that you have following data in **"testout.txt"** file:

```
javaTpoint
```

Output:

```
javaTpoint
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

BUFFEREDOUTPUTSTREAM

- Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.
- For adding the buffer in an OutputStream, use the BufferedOutputStream class. Let's see the syntax for adding the buffer in an OutputStream:
  - OutputStream os= **new** BufferedOutputStream(**new** FileOutputStream("D:\\IO Package\\testout.txt"));
- Java BufferedOutputStream class declaration
  - Let's see the declaration for Java.io.BufferedOutputStream class:
  - **public class** BufferedOutputStream **extends** FilterOutputStream

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Java BufferedOutputStream class methods

Method	Description
void write(int b)	It writes the specified byte to the buffered output stream.
void write(byte[] b, int off, int len)	It write the bytes from the specified byte-input stream into a specified byte array, starting with the given offset
void flush()	It flushes the buffered output stream.

```java
package com.javatpoint;
import java.io.*;
public class BufferedOutputStreamExample{
public static void main(String args[])throws Exception{
    FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
    BufferedOutputStream bout=new BufferedOutputStream(fout);
    String s="Welcome to javaTpoint.";
    byte b[]=s.getBytes();
    bout.write(b);
    bout.flush();
    bout.close();
    fout.close();
    System.out.println("success");
}
}
```

Output:

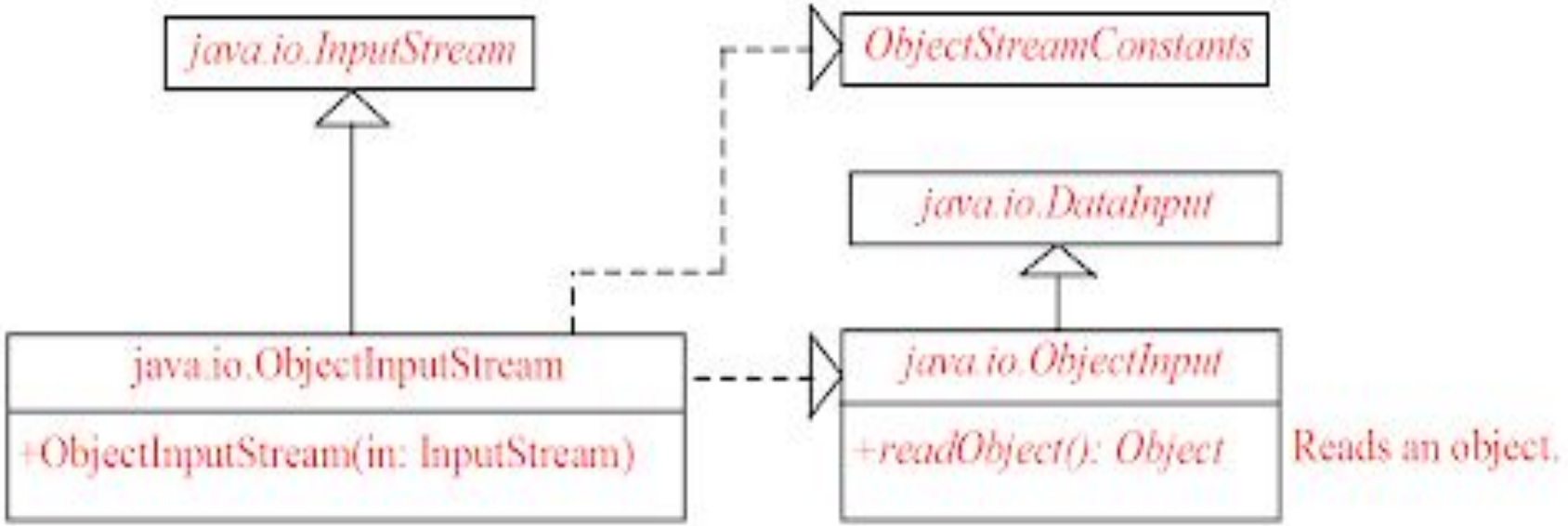
```
Success
```

testout.txt

```
Welcome to javaTpoint.
```

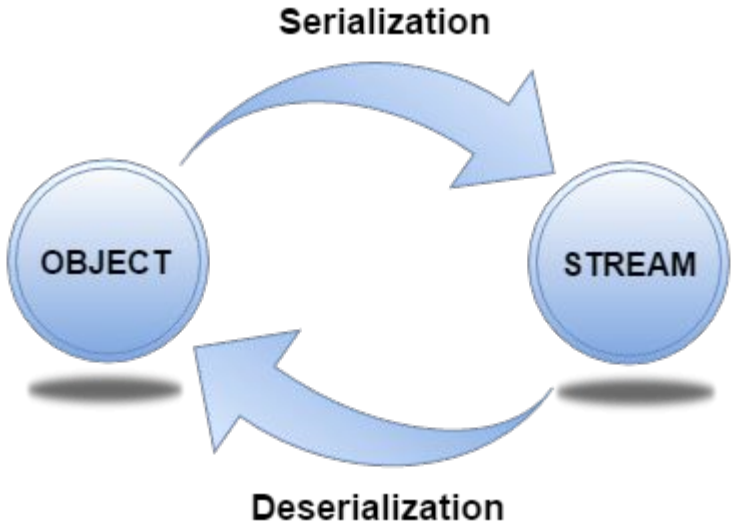
OBJECT I/O

- The object I/O supports ObjectInputStream and ObjectOutputStream classes.These classes to perform I/O operations for object in addition to primitive data types.
- The ObjectInputStream is a subclass of InputStream and implements ObjectInput and ObjectStramConstant.
- Public ObjectInputStream(InputStream in)
- public ObjectOutputStream(OutputStream out)



THE SERIALIZABLE INTERFACE

- Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The Cloneable and Remote are also marker interfaces.
- It must be implemented by the class whose object you want to persist.
- The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

RANDOM ACCESS FILES

- This class is used for reading and writing to randomaccessfile. A random access file behaves like a large array of bytes. There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Modifier and Type	Method	Method
void	close()	It closes this random access file stream and releases any system resources associated with the stream.
FileChannel	getChannel()	It returns the unique FileChannel object associated with this file.
int	readInt()	It reads a signed 32-bit integer from this file.
String	readUTF()	It reads in a string from this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
void	writeDouble(double v)	It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.
void	writeFloat(float v)	It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
void	write(int b)	It writes the specified byte to this file.
int	read()	It reads a byte of data from this file.
long	length()	It returns the length of this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.

```java
import java.io.IOException;
import java.io.RandomAccessFile;

public class RandomAccessFileExample {
   static final String FILEPATH ="myFile.TXT";
   public static void main(String[] args) {
      try {
         System.out.println(new String(readFromFile(FILEPATH, 0, 18)));
         writeToFile(FILEPATH, "I love my country and my people", 31);
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
   private static byte[] readFromFile(String filePath, int position, int size)
         throws IOException {
      RandomAccessFile file = new RandomAccessFile(filePath, "r");
      file.seek(position);
      byte[] bytes = new byte[size];
      file.read(bytes);
      file.close();
      return bytes;
   }
   private static void writeToFile(String filePath, String data, int position)
         throws IOException {
      RandomAccessFile file = new RandomAccessFile(filePath, "rw");
      file.seek(position);
      file.write(data.getBytes());
      file.close();
   }
}
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

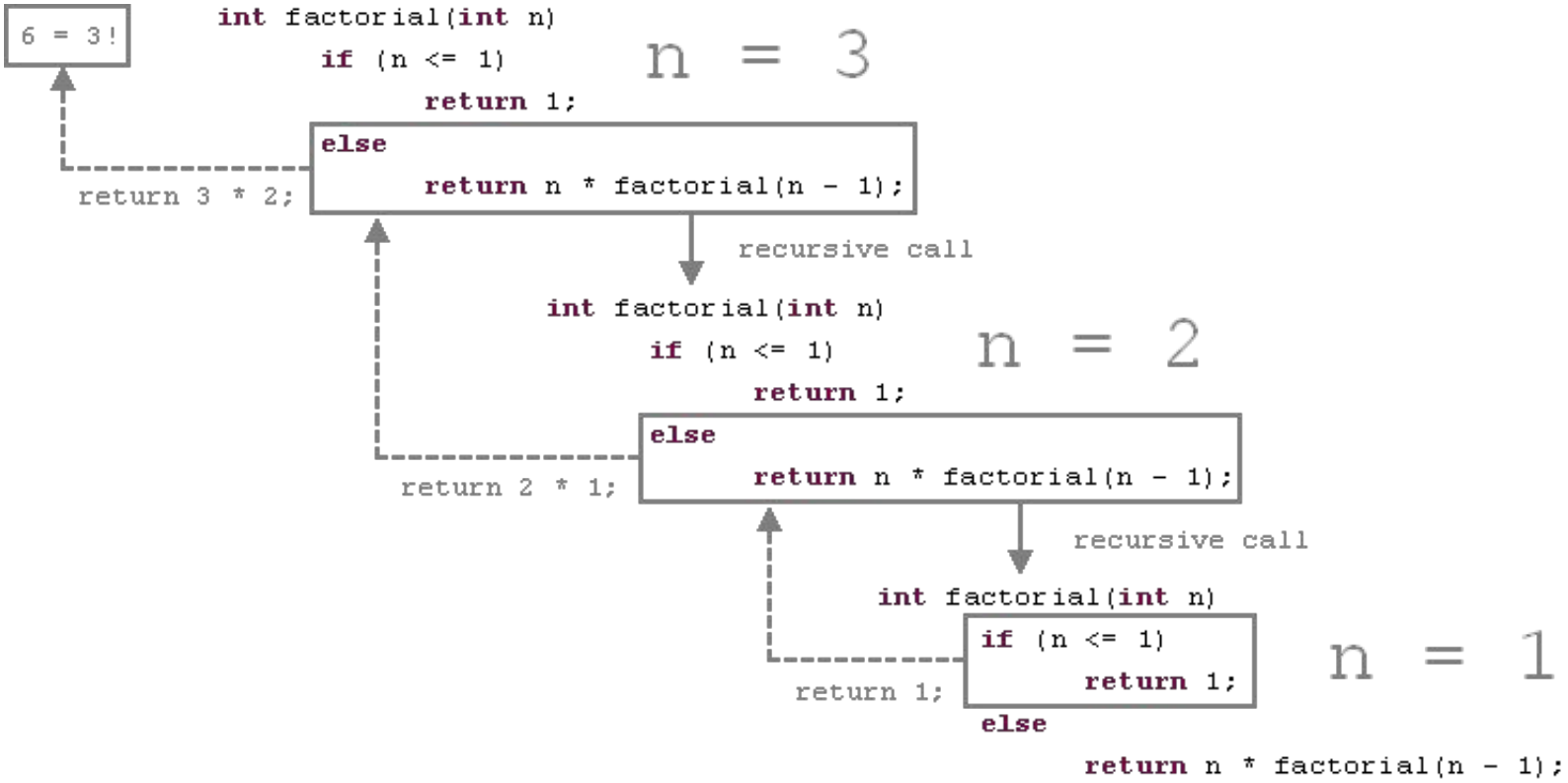
RECURSION

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

PROBLEM SOLVING USING RECURSION

- Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.
- It makes the code compact but complex to understand.
- **Syntax:**
  - returntype methodname(){
  - //code to be executed
  - methodname();//calling same method
  - }

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

RECURSIVE HELPER METHODS

1. we need one helper method where we will pass original string , prefix and one list for result.
2. we will use recursion here. and base case is string is null, in that case we will be returning prefix+orginal string.
3. initially prefix is null and here we will check the 1st character of string is character or not if character we remove that character from original string and will add the same character to prefix and will call uppercase and lowercase method.

TAIL RECURSION

➔ A **tail-recursive** function is just a function whose very the last action is a **call** to itself. **Tail-Call** Optimisation(TCO) lets us convert regular **recursive** calls into **tail** calls to make recursions practical for large inputs, which was earlier leading to stack overflow error in normal **recursion** scenario

```
public class TailRecDemo{
    public static void main(String []args)
        {
            fun(3);
        }
        Public static void fun(int n)
        {
        System.out.println(n);
        if(n>0)
        fun(n-1);
        }
    }
}
```

Criteria	Iteration	Recursion
Mode of implementation	Implemented using loops	Function calls itself
State	Defined by the control variable's value	Defined by the parameter values stored in stack
Progression	The value of control variable moves towards the value in condition	The function state converges towards the base case
Termination	Loop ends when control variable's value satisfies the condition	Recursion ends when base case becomes true
Code Size	Iterative code tends to be bigger in size	Recursion decrease the size of code
No Termination State	Infinite Loops uses CPU Cycles	Infinite Recursion may cause Stack Overflow error or it might crash the system
Execution	Execution is faster	Execution is slower

GENERICS

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

What is generic programming?

- ➔ Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.
- ➔ Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.
- ➔ Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements.

What is the need for generic?

- ➔ It saves the programmers burden of creating separate methods for handling data belonging to different data types.
- ➔ It allows the code reusability
- ➔ Compact code can be created.

Defining generic classes and interfaces

```
public class Test<T>
{
   public Test(){val=null;}
   public Test(T val)
    {
        this.val=val;
    }
   public getVal()
    {
        return val;
    }
    public setVal()
    {
        val= newValue;
    }
    private T val; //variable defined as a generics
}
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Generic methods

- *Generic methods* are methods that introduce their own type parameters. This is similar to declaring a generic type, but the type parameter scope is limited to the method where it is declared. Static and non-static generic methods are allowed, as well as generic class constructors.
- The syntax for a generic method includes a list of type parameters, inside angle brackets, which appears before the method's return type. For static generic methods, the type parameter section must appear before the method's return type.

```java
package javaapplicationoverloadprog1;
import java.io.*;
import java.util.*;
public class JavaApplicationOverloadProg1 {

    public static void display(float[]a)
    {
        for(int i=0;i<5;i++)
            System.out.printf("%f",a[i]);
    }

    public static void display(int[]a)
    {
        for(int i=0;i<5;i++)
            System.out.printf("%d",a[i]);
    }

    public static void display(char[]a)
    {
        for(int i=0;i<5;i++)
            System.out.printf("%c",a[i]);
    }
```

```
    public static void main(String[] args) {
        float[] dbl_a={11,22,33,44,55};
        int[] int_a={1,2,3,4,5};
        char[]char_a={'A','B','C','D','E'};
        System.out.println("\n The Float elements are:");
        display(dbl_a);
        System.out.println("\n The int elements are:");
        display(int_a);
        System.out.println("\n The char elements are:");
        display(char_a);
    }

}
```

- JavaApplicationOverloadProg1 (run)

```
run:

 The Float elements are:
11.00000022.00000033.00000044.00000055.000000
 The int elements are:
12345
 The char elements are:
ABCDEBUILD SUCCESSFUL (total time: 0 seconds)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

```java
package javaapplicationoverloadprog1;
import java.io.*;
import java.util.*;
public class JavaApplicationOverloadProg1 {

     public static <T> void display(T[]a)
         {
             for(int i=0;i<5;i++)
                 System.out.printf("%s",a[i]);
         }
    public static void main(String[] args) {
       float[] dbl_a={11,22,33,44,55};
       int[] int_a={1,2,3,4,5};
       char[]char_a={'A','B','C','D','E'};
       System.out.println("\n The Float elements are:");
       display(dbl_a);
       System.out.println("\n The int elements are:");
       display(int_a);
       System.out.println("\n The char elements are:");
       display(char_a);
    }
```

```
t - JavaApplicationOverloadProg1 (run) ×
run:

 The Float elements are:
11.00000022.00000033.00000044.00000055.000000
 The int elements are:
12345
 The char elements are:
ABCDEBUILD SUCCESSFUL (total time: 0 seconds)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Raw types and backward compatibility

- ➔ Generic class or interface is used without specifying a concrete type.
- ➔ It enables backward compatibility with earlier versions of java
- ➔ Example:

  GenericsStack stack= new GenericsStack();

  This is almost equivalent to

GenericsStack <Object> stack = new GenericsStack<Object>();

- ➔ The raw types are unsafe.

Concept of bounded type

- There may be times when you want to restrict the types that can be used as type arguments in a parameterized type. For example, a method that operates on numbers might only want to accept instances of Number or its subclasses. This is what bounded type parameters are for.
  - Sometimes we don’t want whole class to be parameterized, in that case we can create java generics method. Since constructor is a special kind of method, we can use generics type in constructors too.
  - Suppose we want to restrict the type of objects that can be used in the parameterized type. For example in a method that compares two objects and we want to make sure that the accepted objects are Comparables.
  - The invocation of these methods is similar to unbounded method except that if we will try to use any class that is not Comparable, it will throw compile time error.

```java
package javaapplicationboundedtypedemo;
public class JavaApplicationBoundedTypeDemo {

    class Test <T extends Number>
    {
        T t;
        public Test(T t)
        {
            this.t = t;
        }
        public T getT()
        {
            return t;
        }
    }
    public static void main(String[] args)
    {
       Test<Number>obj1 = new Test<Number>(123);
       System.out.println("The intger is:"+obj1.getT());
       Test<String>obj2 = new Test<String>("I am String");
       System.out.println("The string is:"+obj2.getT());
    }

}
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Wildcard generic types

- The question mark (?) is known as the wildcard in generic programming . It represents an unknown type. The wildcard can be used in a variety of situations such as the type of a parameter, field, or local variable; sometimes as a return type. Unlike arrays, different instantiations of a generic type are not compatible with each other, not even explicitly. This incompatibility may be softened by the wildcard if ? is used as an actual type parameter.
- There are three ways to use wildcards

1. Unbounded wildcard
2. Upper bound wildcard
3. Lower bound wildcard

Unbounded wildcard

- These wildcards can be used when you want to relax the restrictions on a variable. For example, say you want to write a method that works on List < integer >, List < double >, and List < number > , you can do this using an upper bounded wildcard.
- To declare an upper-bounded wildcard, use the wildcard character (‘?’), followed by the extends keyword, followed by its upper bound.

```
package javaapplicationunboundedcarddemo;

import java.util.Arrays;
import java.util.List;

public class JavaApplicationUnboundedCardDemo {


    public static void main(String[] args) {
        List<Integer> list1= Arrays.asList(10,20,30,40);

        List<String> list2 = Arrays.asList("AAA","BBB","CCC");
        Display(list1);
        Display(list2);
    }

    private static void Display(List<?> mylist)
    {
        System.out.println(mylist);
    }
```

```
ut - JavaApplicationUnboundedCardDemo (run)
run:
[10, 20, 30, 40]
[AAA, BBB, CCC]
BUILD SUCCESSFUL (total time: 0 seconds)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Upper Bound wildcard

- This wildcard type is specified using the wildcard character (?), for example, List. This is called a list of unknown type. These are useful in the following cases
  - When writing a method which can be employed using functionality provided in Object class.
  - When the code is using methods in the generic class that don't depend on the type parameter

```
package javaapplicationunboundedcarddemo;

import java.util.Arrays;
import java.util.List;
public class JavaApplicationUnboundedCardDemo {
    public static void main(String[] args) {
        //Integer List
         List<Integer> list1= Arrays.asList(10,20,30,40);
         //Double list
         List<Double> list2=Arrays.asList(11.11,22.22,33.33,44.44);

        System.out.println(sum(list1));
        System.out.println(sum(list2));
    }
    private static double sum(List<? extends Number> mylist)
    {

        double total=0.0;
        for(Number i:mylist)
        {
            total+=i.doubleValue();
        }
        return total;
    }
}
```

```
t - JavaApplicationUnboundedCardDemo (run)
run:
100.0
111.1
BUILD SUCCESSFUL (total time: 0 seconds)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Lower Bound wildcard

➔ It is expressed using the wildcard character (‘?’), followed by the super keyword, followed by its lower bound: <? super A>.

◆ **Syntax:**

Collectiontype <? super A>

```
package javaapplicationunboundedcarddemo;

import java.util.Arrays;
import java.util.List;
public class JavaApplicationUnboundedCardDemo {
    public static void main(String[] args) {
        //Lower Bounded Integer List
        List<Integer> list1= Arrays.asList(4,5,6,7);

        //Integer list object is being passed
        printOnlyIntegerClassorSuperClass(list1);

        //Number list
        List<Number> list2= Arrays.asList(4,5,6,7);

        //Integer list object is being passed
        printOnlyIntegerClassorSuperClass(list2);
    }

    public static void printOnlyIntegerClassorSuperClass(List<? super Integer> list)
    {
        System.out.println(list);
    }
}
```

```
t - JavaApplicationUnboundedCardDemo (run) ×
run:
[4, 5, 6, 7]
[4, 5, 6, 7]
BUILD SUCCESSFUL (total time: 0 seconds)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Erasure and restrictions on generics

- Type Erasure rules
  - Replace type parameters in generic type with their bound if bounded type parameters are used.
  - Replace type parameters in generic type with Object if unbounded type parameters are used.
  - Insert type casts to preserve type safety.
  - Generate bridge methods to keep polymorphism in extended generic types.

Restrictions on generics

1. It cannot instantiate Generic Types with primitive data types.
2. It cannot create instance of Type Parameter
3. It cannot declare static field whose types are types parameter
4. It cannot use cast or instances of operators with parameterized types.
5. It cannot create Array of parameterized types.
6. It cannot create,catch or throw Objects of parameterized types.
7. It cannot overload a method where the formal parameter Types of Each Overloaded Erase to the same Raw Type.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY