Core Java

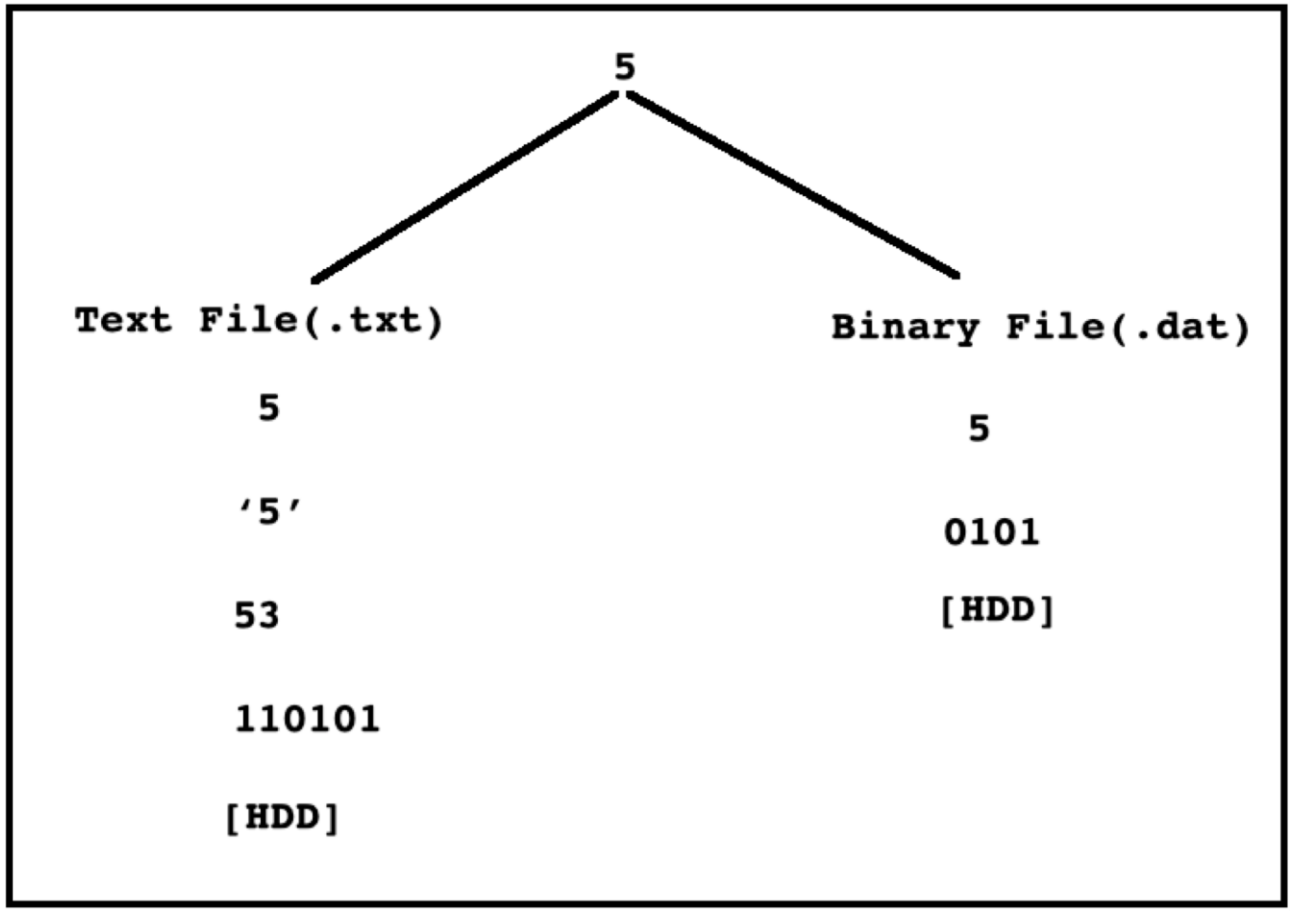
File Handling

Akshita Chanchlani

Introduction

- **Variable**
  - It is temporary container, which is used to store data on RAM.
- **File**
  - It is permanent container, which is used to store data on HDD.
- File is operating system resource/non java resource.
- General types of file:
  1. Text File
  2. Binary File

Text File versus Binary File



Text File

- .txt, .rtf, .doc/.docx, .html/.css/.js/.xml, .c/cpp/.java etc.
- We can read text file using any text editor(example notepad).
- It contains data in human readable format.
- It requires more processing hence it is slower in performance.

Binary File

- .jpg/.jpeg/.png, .mp3/.mp4, .o/.obj, .class etc.
- To read binary file we need to use specific program.
- It doesn't contains data in human readable format.
- It requires less processing hence it is faster in performance.

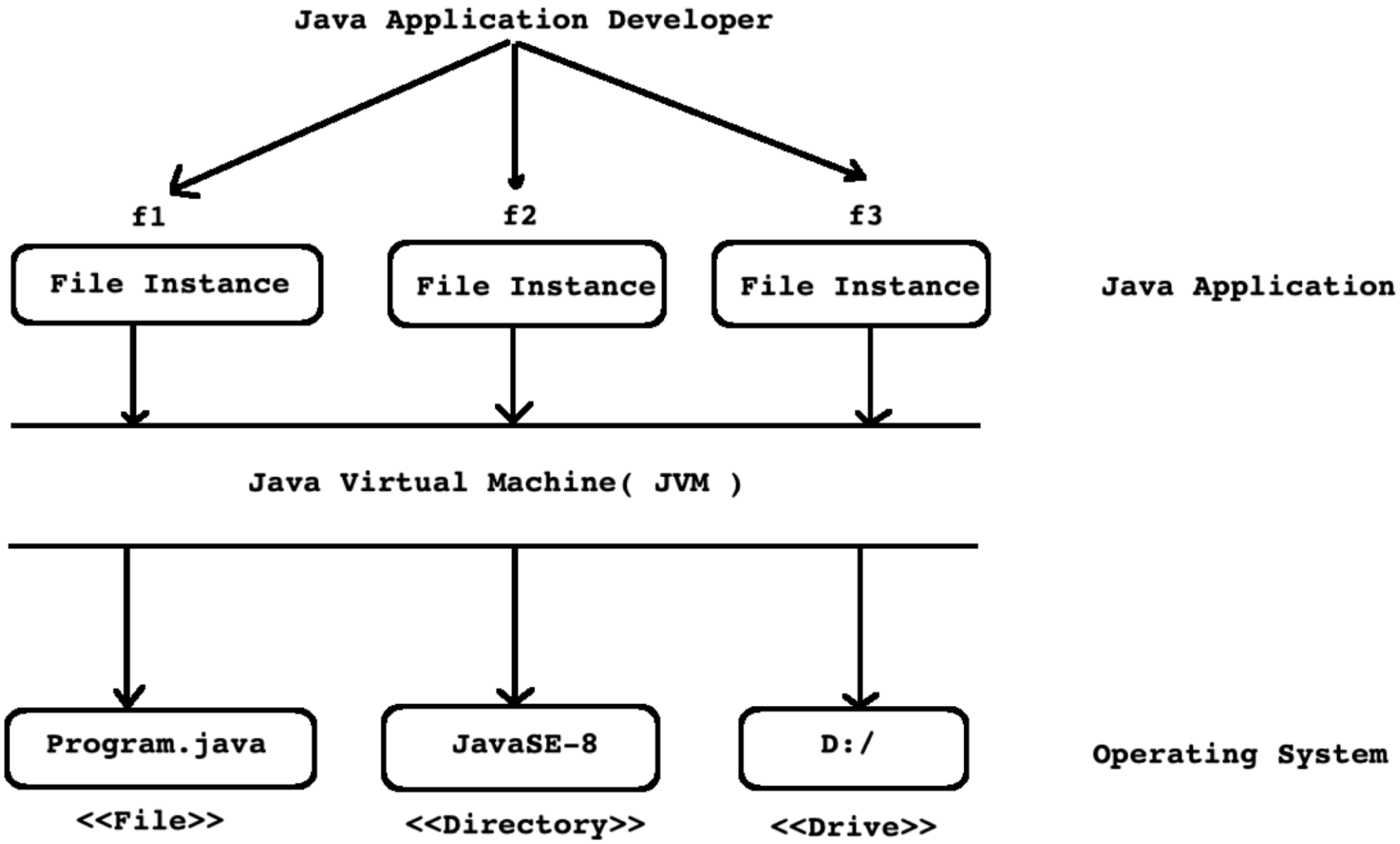
Absolute Path versus Relative Path

- A path of a file from root directory is called as absolute path.
  - Example : **D:\JavaSE8\Demo\src\Program.java**
- A path of a file from current directory is called as relative path.
  - Example : **./src/Program.java**
- Path represents location of file/directory on HDD. It contains:
  1. Root directory name
  2. Sub directories
  3. File Name
  4. Path separator character
  5. Windows : \
  6. Unix\Linux : /

File Handling In Java.

- If we want to manage file in Java then we should use types declared in **java.io and java.nio** package.
- Console is a class declared in java.io package which represents console/terminal associated with JVM
- File is a class declared in java.io package which represents Operating system Drive/directory/file.
- We should use File class:
    1. To create new empty file
    2. To create new empty directory
    3. To delete existing file/directory
    4. To read metadata of file/folder or directory/drive

File Handling In Java.

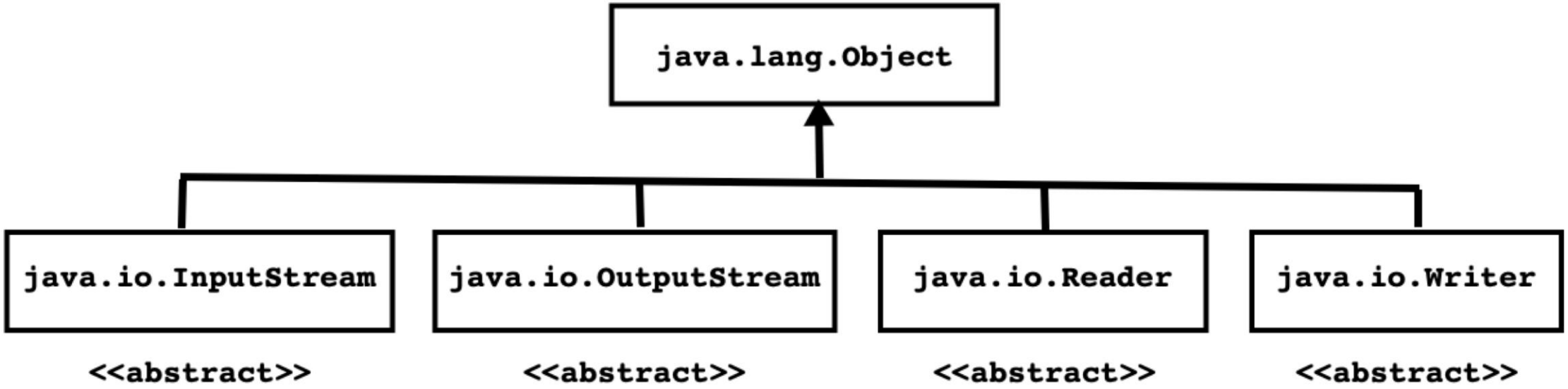


Methods of java.io.File class

1. To create new empty file
   - public boolean **createNewFile**() throws IOException
2. To create new empty directory
   - public boolean **mkdir**()
3. To delete existing file/directory
   - public boolean **delete**()
4. To read metadata of file/directory/drive
   - public String getName()
   - public String getParent()
   - public String getPath()
   - public long length()
   - public String[] list()
   - public File[] listFiles()
   - public long getTotalSpace()
   - public long getUsableSpace()
   - public long getFreeSpace()
   - public boolean isFile()
   - public boolean isDirectory()
   - public boolean isHidden()

File I/O

- If we want to read/write data to and from file then we should use stream.



- Stream is an abstraction(instance/object) which either consume(read)/produce(write) information from source to destination.
- InputStream, OutputStream and their sub classes are used to process binary file.
- Reader, Writer and their sub classes are used to process text file.
- Since all these classes implements Closeable interface, we can use its instance in try with resource syntax.

OutputStream Hierarchy

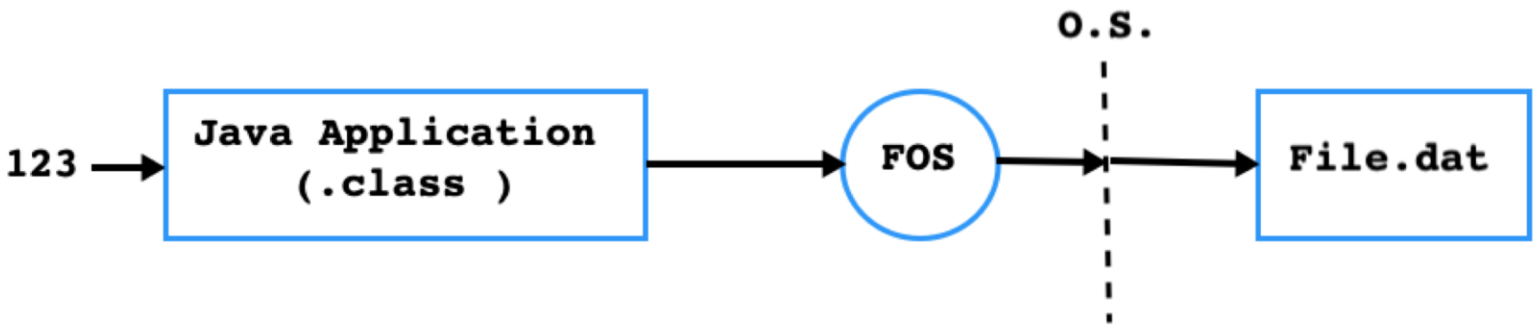
- java.lang.object
  - o java.io.outputStream (implements java.io.Closeable, java.io.Flushable)
    - ❑ java.io.FileoutputStream
    - ❑ java.io.FilteroutputStream
      - ➢ java.io.BufferedoutputStream
      - ➢ java.io.DataoutputStream (implements java.io.Dataoutput)
      - ➢ java.io.PrintStream (implements java.lang.Appendable, java.io.Closeable)
    - ❑ java.io.objectoutputStream (implements java.io.objectoutput

InputStream Hierarchy

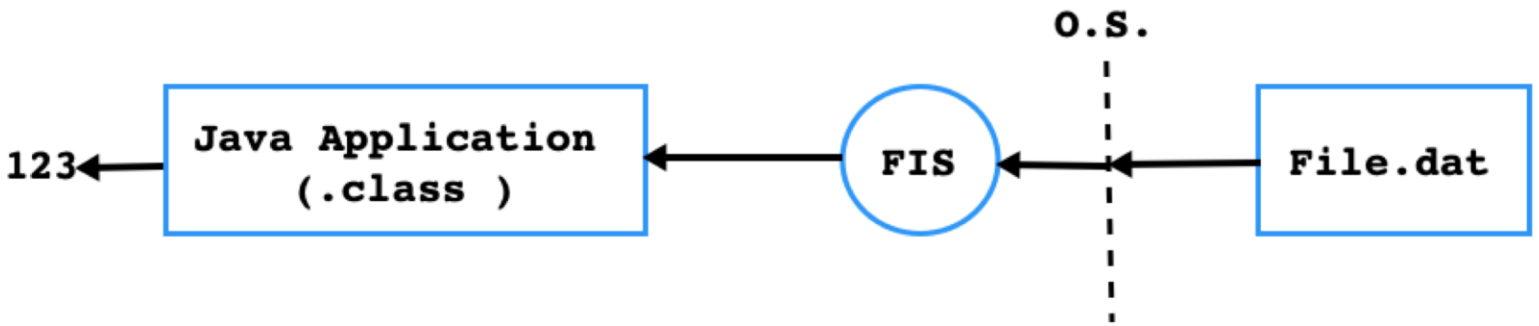
- java.lang.object
  - java.io.InputStream (implements java.io.Closeable)
    - java.io.FileInputStream
    - java.io.FilterInputStream
      - java.io.BufferedInputStream
      - java.io.DataInputStream (implements java.io.DataInput)
    - java.io.objectInputStream (implements java.io.objectInput)

FileInputStream and FileOutputStream

- Using FileOutputStream Instance, we can write single byte at a time into binary file.

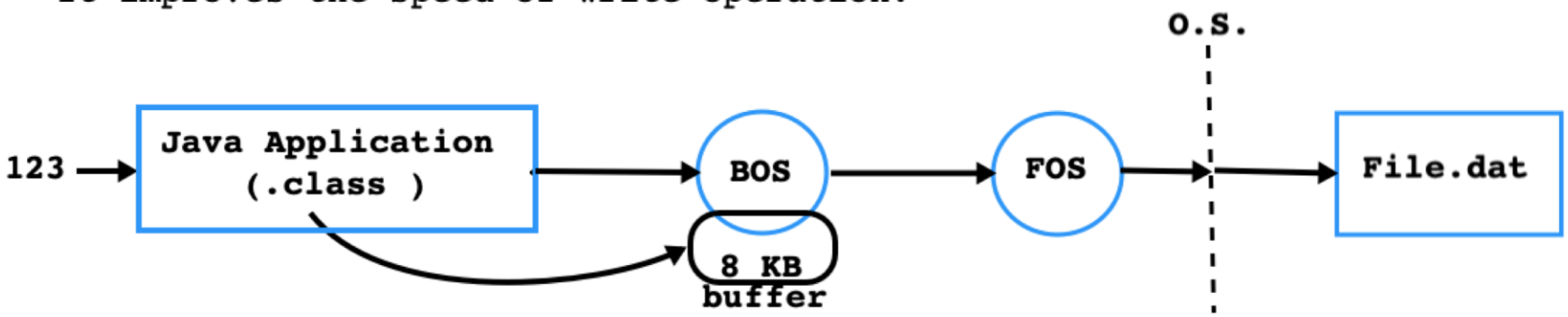


- Using FileInputStream Instance, we can read single byte at a time from binary file.

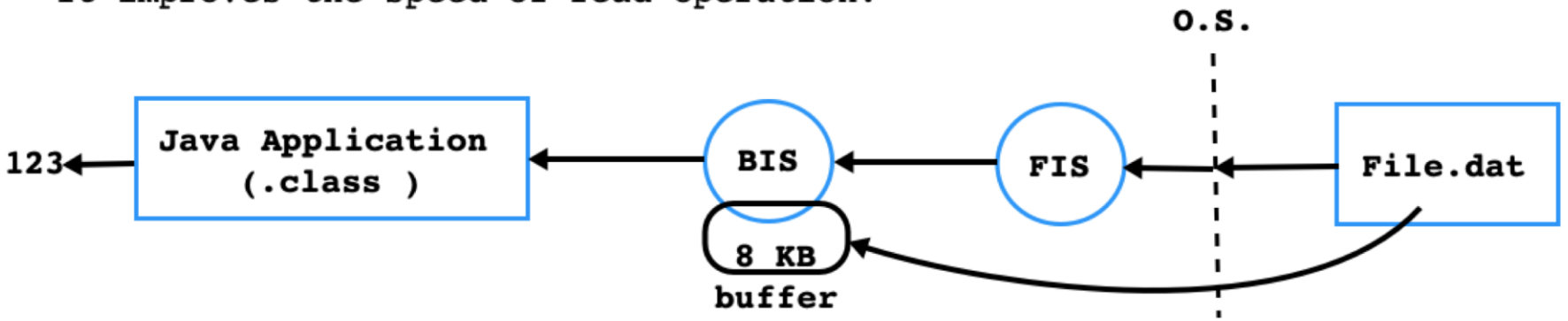


BufferedInputStream and BufferedOutputStream

- Using BufferedOutputStream Instance, we can write multiple bytes at a time into binary file.
- It improves the speed of write operation.

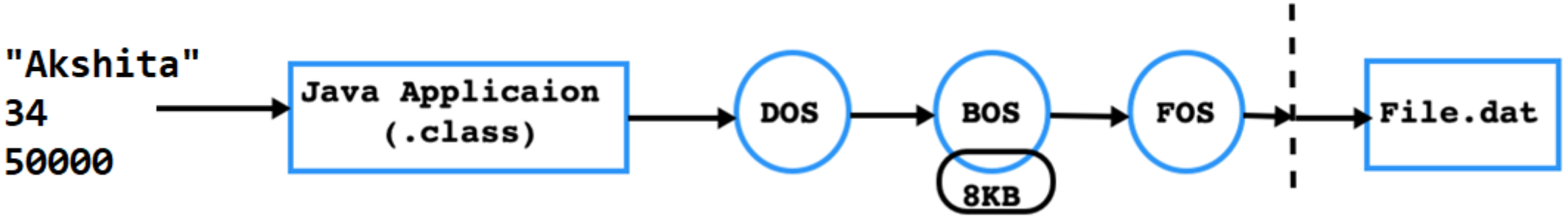


- Using BufferedInputStream Instance, we can read multiple bytes at a time from binary file.
- It improves the speed of read operation.

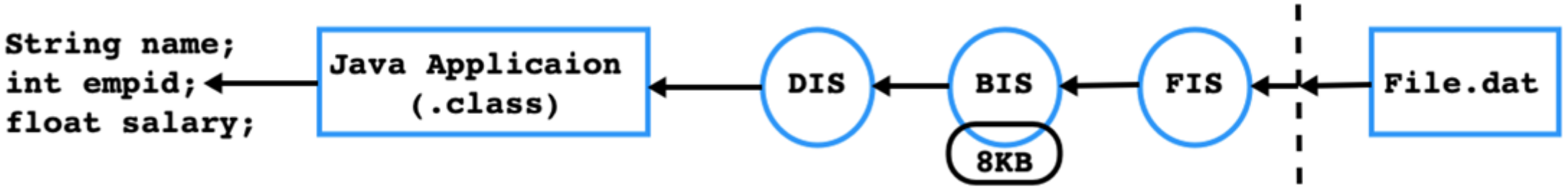


DataInputStream and DataOutputStream

- DataOutputStream instance is used to convert primitive values into binary data.

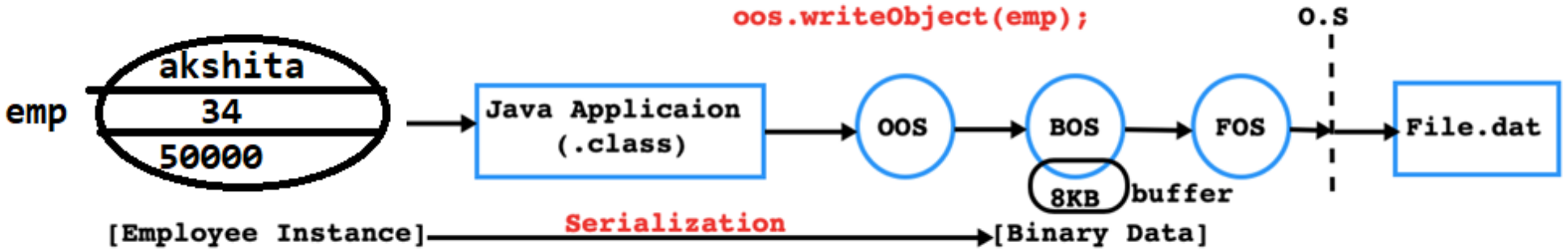


- DataInputStream instance is used to convert binary data into primitive values.

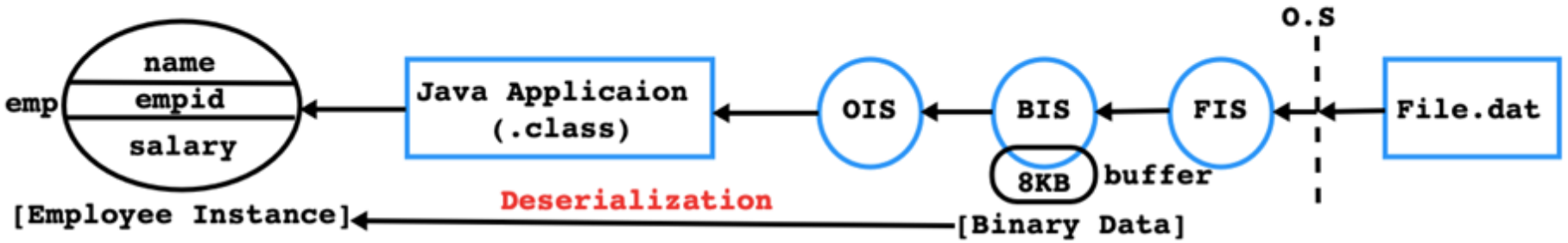


ObjectInputStream and ObjectOutputStream

- ObjectOutputStream instance is used to convert State of Java Instance into binary data.



- ObjectInputStream instance is used to convert binary into Java Instance .




```
emp = (Employee) ois.readObject( );
```

Serialization

- ObjectOuput is a subinterface of DataOutput interface.
  - **"void writeObject(Object obj)throws IOException"** is a method of ObjectOutput interface.
- ObjectOutputStream class implements ObjectOuput interface.
- If we want to convert state of instance of class into binary data then we should use ObjectOuputStream class.
- This ***process of converting state of Java instance into binary data is called as Serialization***.
- If we want to serialize state of Java instance then its type/class must implement java.io.Serializable interface. Otherwise writeObject() method will throw NotSerializableException.
- Serializable is a marker interface.

Serialization

- In case of association, type of contained instance must also implement Serializable interface.

```
class Date implements Serializable{
   //TODO : Declare fields and methods
}
class Employee implements Serializable{
    private Date joinDate;  //Date must implement Serializable interface.
   //TODO : constructor, getter and setter
}
```

- transient is a keyword in Java.
- If we declare any field transient then JVM do not serialize its state.
- JVM do not serialize state of static and transient field.

Deserialization

- objectInput is a subinterface of DataInput interface.
  - **"Object readObject() throws ClassNotFoundException,IOException"** is a method of objectInput interface.
- objectInputStream class implements objectInput interface.
- If we want to convert binary data into state of instance of class then we should use objectInputStream class.
- This ***process of converting binary data into state of Java instance is called as deserialization***.

SerialVersionUID

- The serialization runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization. If the receiver has loaded a class for the object that has a different serialVersionUID than that of the corresponding sender's class, then deserialization will result in an InvalidClassException.
- A serializable class can declare its own serialVersionUID explicitly by declaring a field named "serialVersionUID" that must be static, final, and of type long:
  - **ANY-ACCESS-MoDIFIER static final long serialVersionUID = 42L;**

SerialVersionUID

- If a serializable class does not explicitly declare a serialVersionUID, then the serialization runtime will calculate a default serialVersionUID value for that class based on various aspects of the class, as described in the Java(TM) Object Serialization Specification. However, it is strongly recommended that all serializable classes explicitly declare serialVersionUID values

```
class Employee implements Serializable{
    private static final long serialVersionUID = 750431028817745332 1L;
}
```

Writer Hierarchy

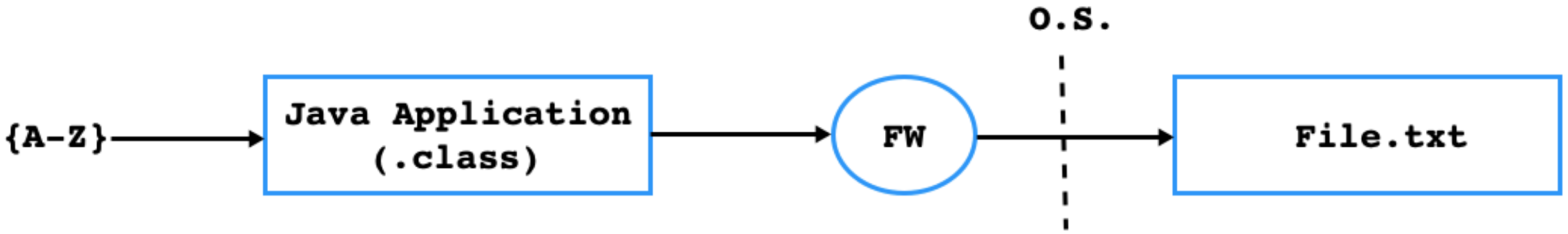
- java.lang.object
  - java.io.Writer (implements java.lang.Appendable, java.io.Closeable, java.io.Flushable)
    - java.io.BufferedWriter
    - java.io.CharArrayWriter
    - java.io.FilterWriter
    - java.io.outputStreamWriter
      - java.io.FileWriter
    - java.io.PipedWriter
    - java.io.PrintWriter
    - java.io.StringWriter

Reader Hierarchy

- java.lang.object
  - java.io.Reader (implements java.io.Closeable, java.lang.Readable)
    - java.io.BufferedReader
      - java.io.LineNumberReader
    - java.io.CharArrayReader
    - java.io.FilterReader
      - java.io.PushbackReader
    - java.io.InputStreamReader
      - java.io.FileReader
    - java.io.PipedReader
    - java.io.StringReader

FileWriter and FileReader

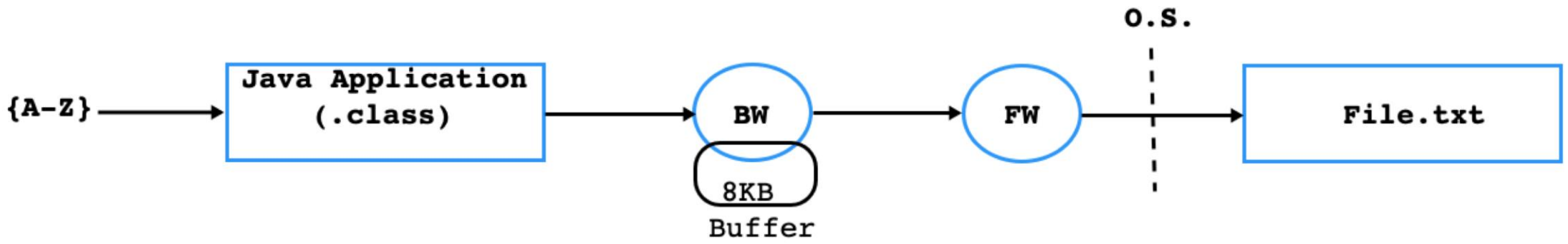
- FileWriter instance is used to write single character at a time inside file.



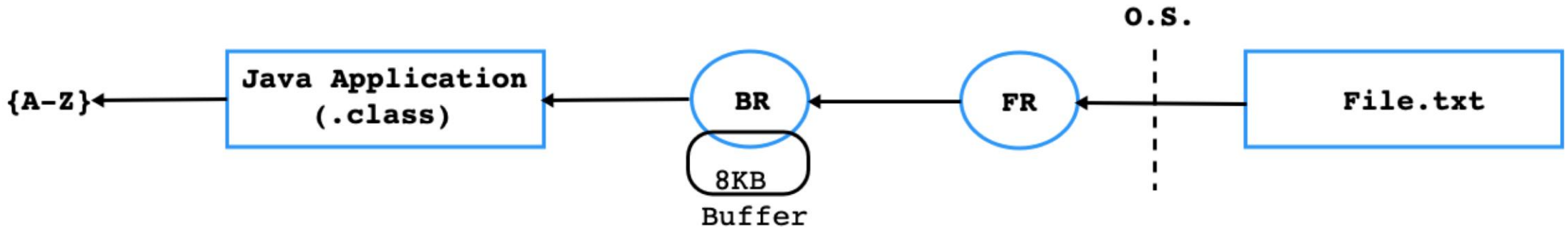
- FileReader instance is used to read single character at a time from file.

BufferedReader and BufferedWriter

- Using BufferedWriter instance, we can write multiple characters inside file.



- Using BufferedReader instance, we can read multiple characters from file.



Thank You.