Practical File

Big Data -1 (BCA-DS-305)

An Initiative by "The Last Centre"

ECHELON

TRANSFORMING THE EDUCATION

Submitted To	Submitted By
Mr. Saransh	**Ruchi Singh**
Assistant Professor	**22-BCADS-040**

Department of Computer Science and Engineering

INDEX

Installation of Hadoop

Objective:

To install and configure a Hadoop single-node cluster on a Windows operating system for learning and development purposes.

Prerequisites:

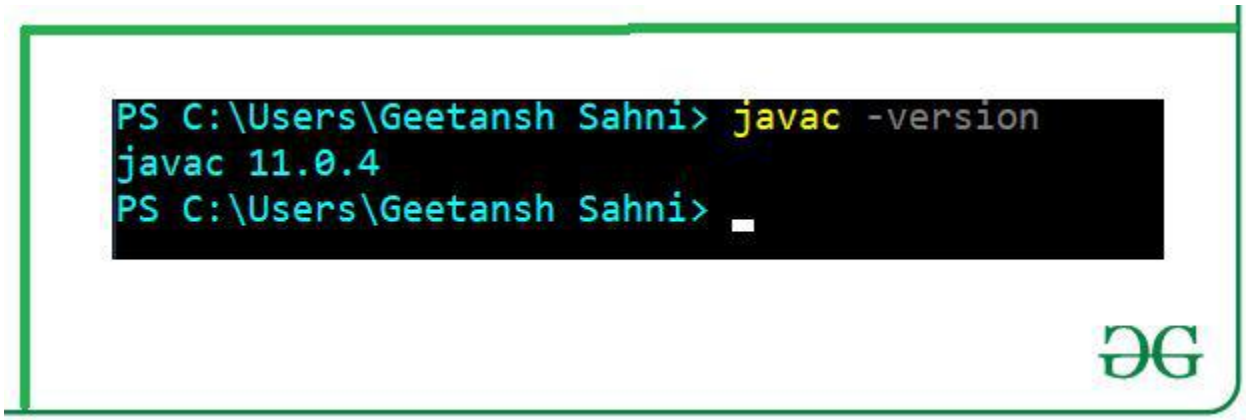
1. **Java Development Kit (JDK):** Ensure Java is installed on your system. Preferably use JDK 8 or above.
2. **Hadoop Package:** Download the stable Hadoop distribution from the Apache Hadoop website.
3. **System Requirements:** Minimum 4GB of RAM and 20GB of free disk space.

Steps for Installation:

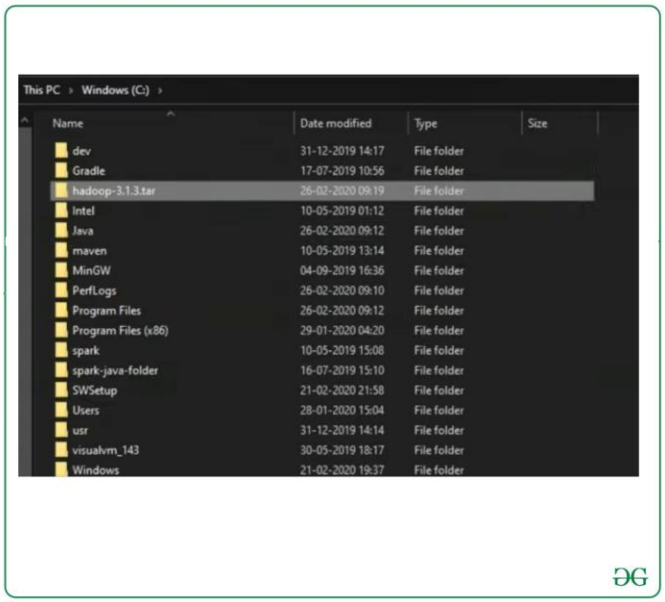
Step 1: Verify Java Installation

```
javac -version
```

If Java is installed, it will display the version number. If not, download and install Java from the Oracle JDK website.

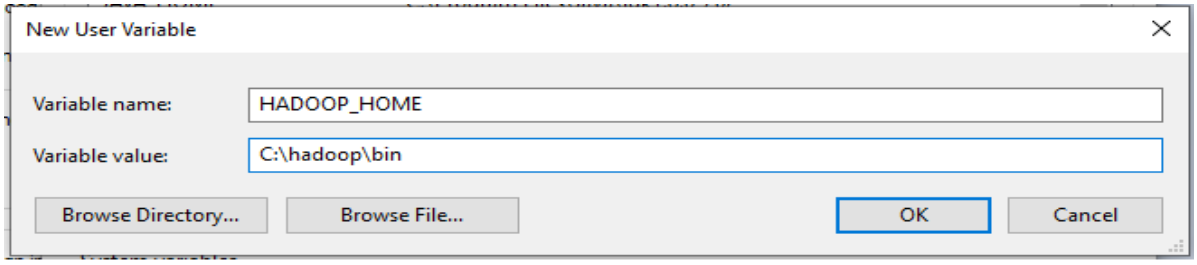


Step 2: Extract Hadoop



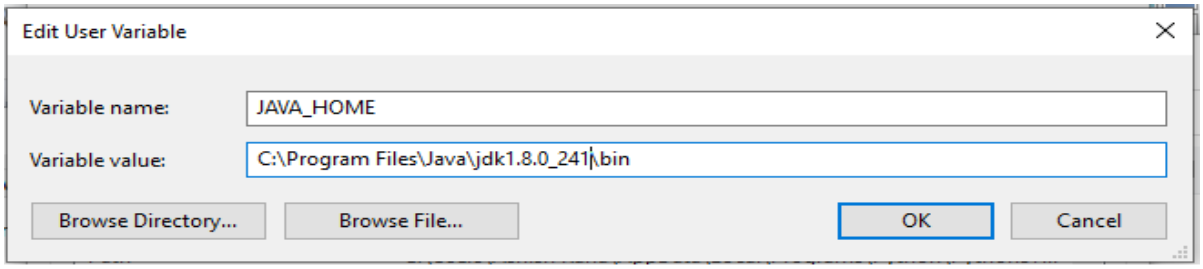
Download the Hadoop package and extract it to `C:\Hadoop` using any extraction tool like WinRAR or 7-Zip.

Step 3: Set `HADOOP_HOME` Environment Variable



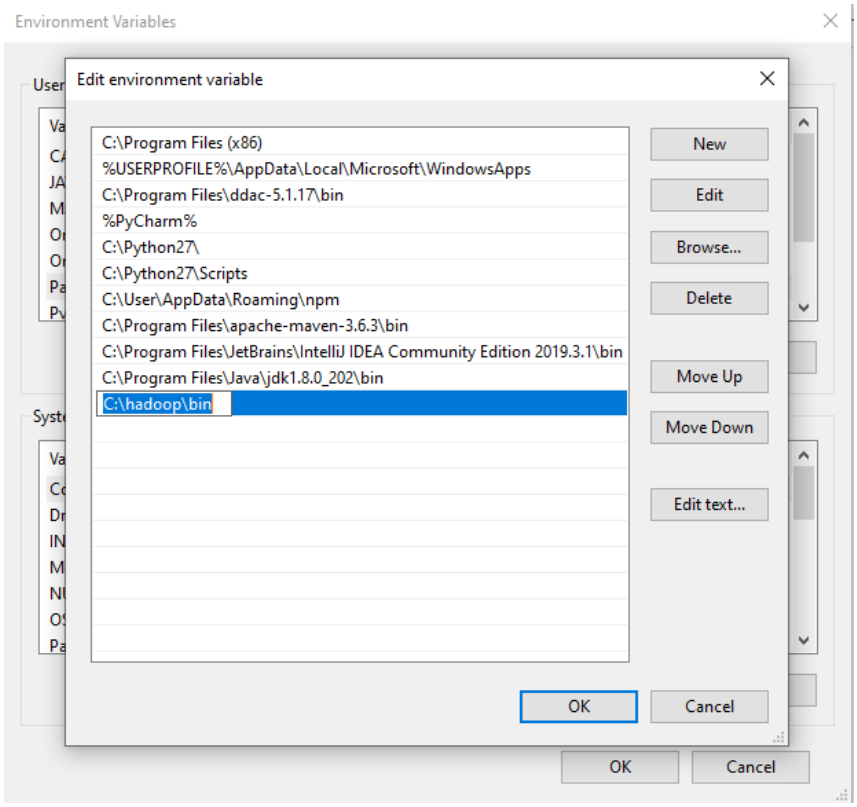
Navigate to the Environment Variables in the system settings, create a new variable named `HADOOP_HOME`, and set its value to `C:/Hadoop`.

Step 4: Set `JAVA_HOME` Environment Variable



Similarly, set another environment variable`JAVA_HOME` pointing to the Java JDK installation directory, such as `C:\Java`.

Step 5: Update the `PATH` Variable



Update the system's Path variable by appending the paths `%HADOOP_HOME%\bin` and `%JAVA_HOME%\bin` , allowing the Command Prompt to recognize Hadoop and Java commands.

Step 6: Hadoop Configuration

For Hadoop Configuration we need to modify Six files that are listed below-

1. Core-site.xml

2. Mapred-site.xml

3. Hdfs-site.xml

4. Yarn-site.xml

5. Hadoop-env.cmd

6. Create two folders datanode and namenode

Step 6.1: `cross-site.xml` Configuration

```
<configuration>
   <property>
       <name>fs.defaultFS</name>
       <value>hdfs://localhost:9000</value>
   </property>
</configuration>
```

Step 6.2: `mapred-site.xml` Configuration

```
<configuration>
   <property>
       <name>mapreduce.framework.name</name>
       <value>yarn</value>
   </property>
</configuration>
```

Step 6.3: `hdfs-site.xml` Configuration

```
<configuration>
   <property>
       <name>dfs.replication</name>
       <value>1</value>
```

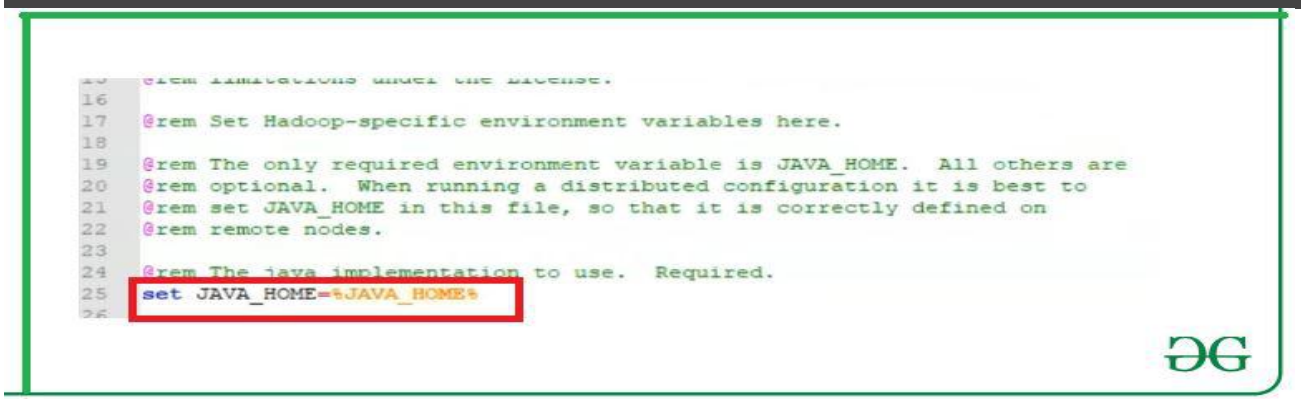
```
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>C:\hadoop-2.8.0\data\namenode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>C:\hadoop-2.8.0\data\datanode</value>
    </property>
</configuration>
```

Step 6.4: `yarn-site.xml` Configuration

```
<configuration>
    <property>
         <name>yarn.nodemanager.aux-services</name>
         <value>mapreduce_shuffle</value>
    </property>
    <property>
           <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
     <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
</configuration>
```

Step 6.5: `hadoop-env.cmd` Configuration

```
Set "JAVA_HOME=C:\Java" (On C:\java this is path to file jdk.18.0)
```

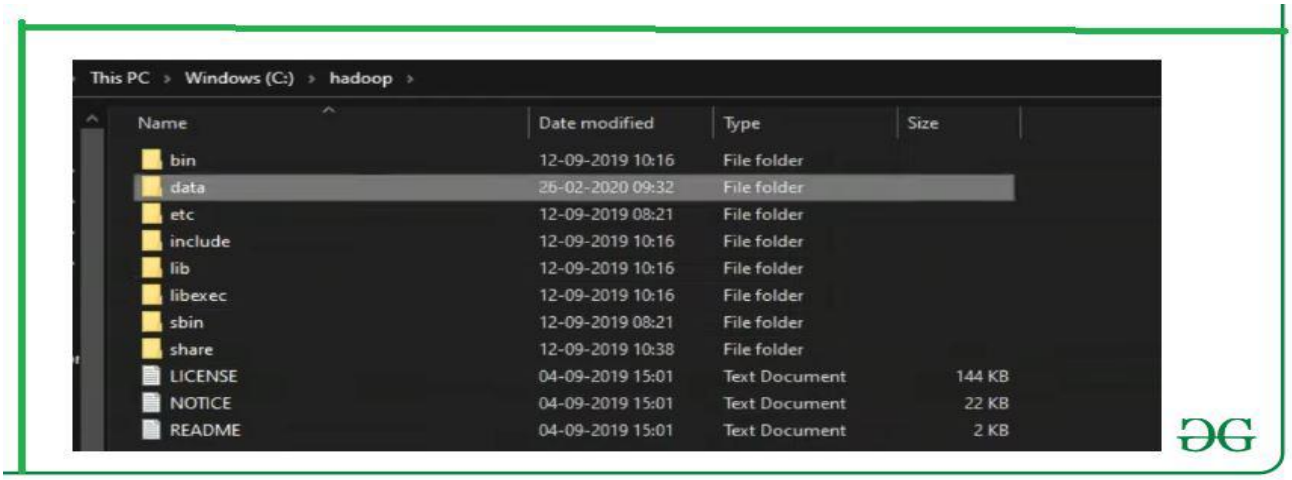


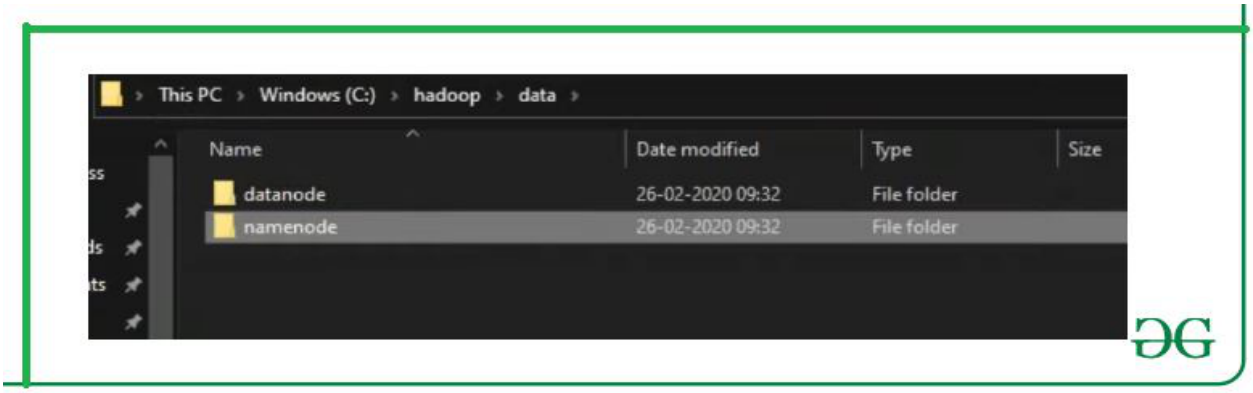
Step 6.6: Create datanode and namenode folders

1. Create folder "data" under "C:\Hadoop-2.8.0"

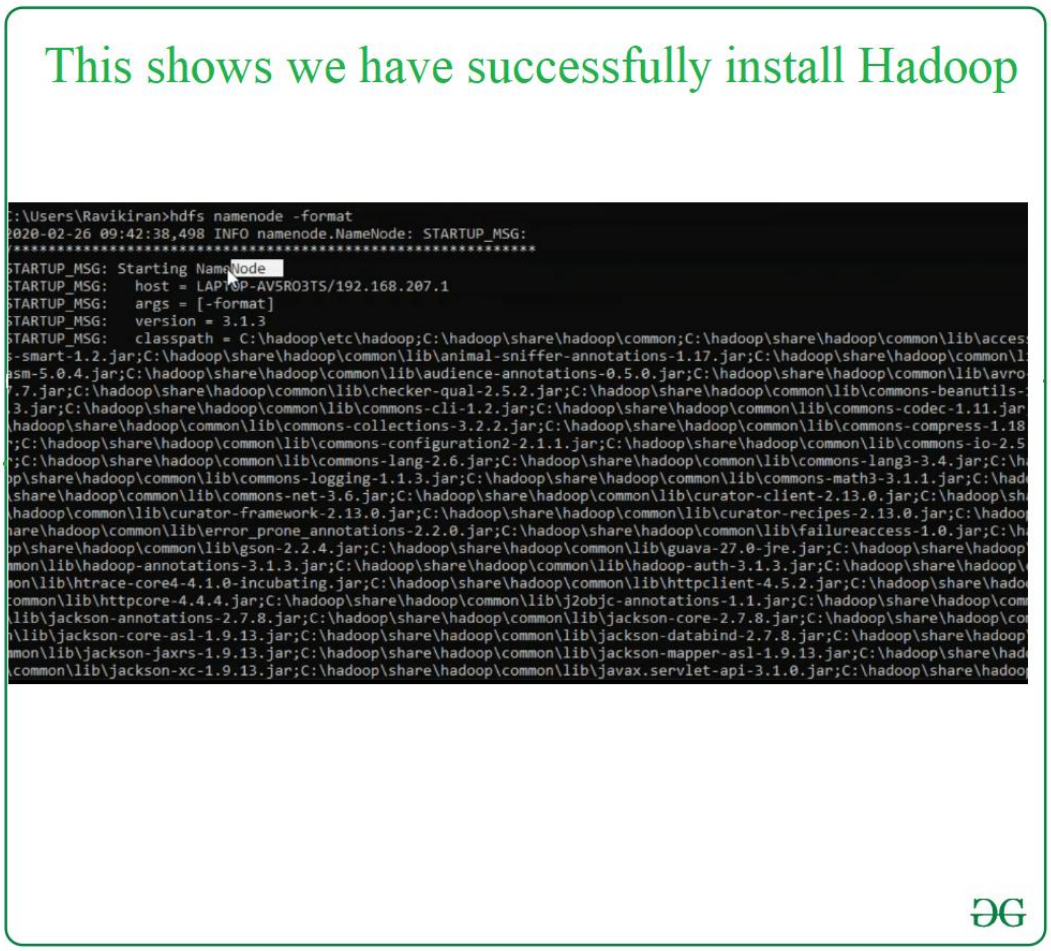
2. Create folder "datanode" under "C:\Hadoop-2.8.0\data"

3. Create folder "namenode" under "C:\Hadoop-2.8.0\data"





Step 7: Format the namenode folder

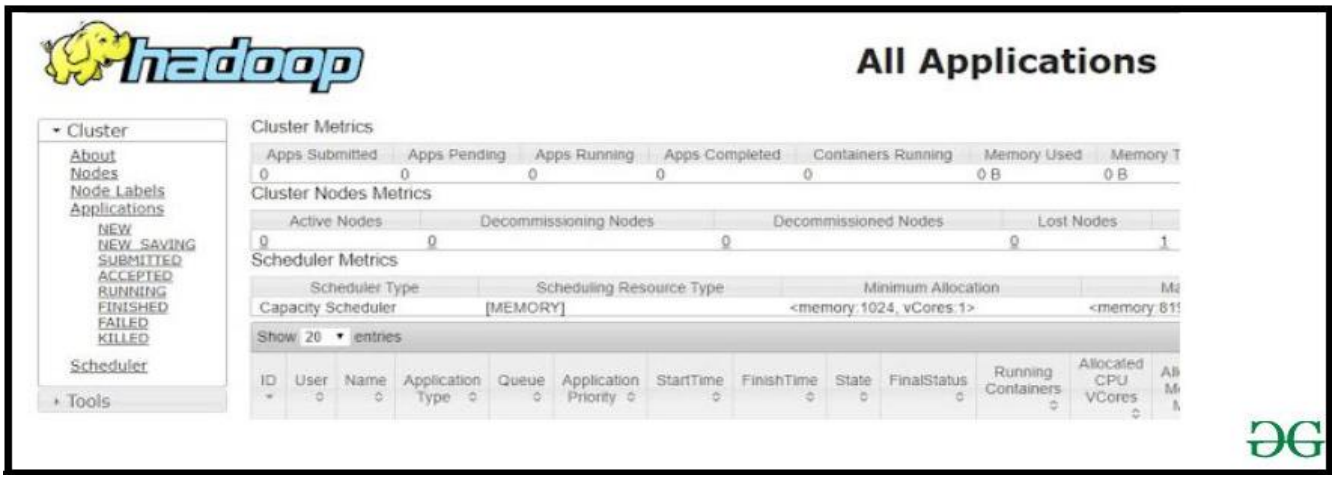


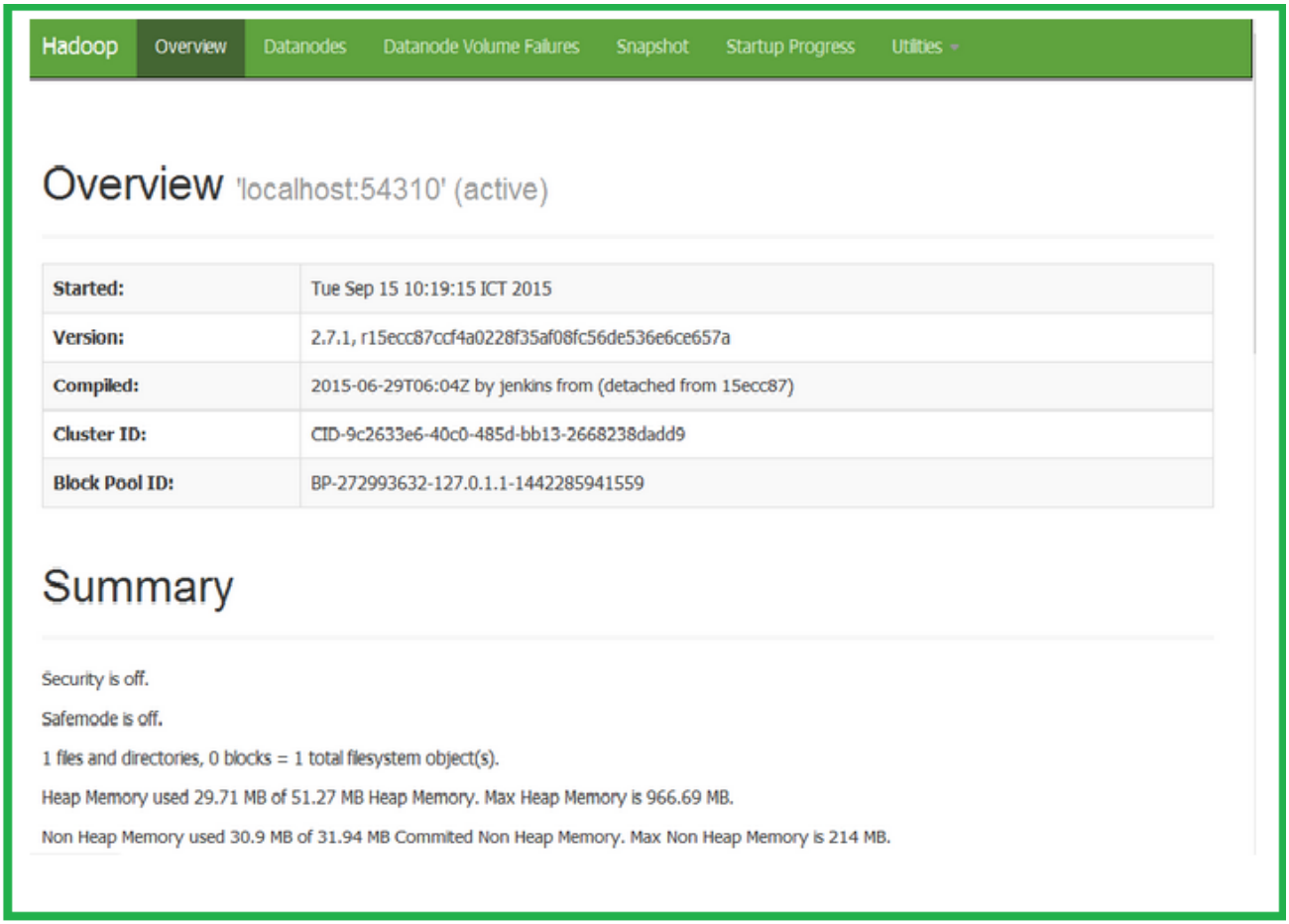
Run the command `hdfs` namenode -format to initialize the Hadoop filesystem and prepare it for use.

Step 8: Start Hadoop Services

Start all Hadoop services by running the command start-all.cmd in the Command Prompt. Alternatively, you can start individual services with `start-dfs.cmd` for HDFS and start-yarn.cmd for YARN.

Step 9: Test Setup





Verify the setup by checking active processes using the `jps` command. Access the Hadoop web interfaces for HDFS at http://localhost:50070 and for YARN Resource Manager at http://localhost:8088.

Conclusion

The installation and configuration of a single-node Hadoop cluster on Windows are successfully completed. This setup allows exploration of Hadoop's features in a local development environment.

Performing Basic HDFS Shell Operations

Objective

To understand and perform basic file system operations in Hadoop Distributed File System (HDFS) using HDFS shell commands.

Commands and Descriptions:

1. **Start Hadoop Services**

   Before running HDFS commands, ensure Hadoop services are started:

```
start-dfs.sh
start-yarn.sh
```

2. **Create a Directory in HDFS**

   Create a directory named `mydir` in the HDFS.

```
hdfs dfs -mkdir /mydir
```

3. **List Files and Directories**

   View the contents of the root directory (/).

```
hdfs dfs -ls /
```

4. **Copy a File from Local Filesystem to HDFS**

   Upload a local file (sample.txt) to the HDFS directory /mydir.

```
hdfs dfs -put /path/to/sample.txt /mydir
```

5. **View the Contents of a File in HDFS**

   Display the contents of sample.txt stored in HDFS.

```
hdfs dfs -cat /mydir/sample.txt
```

6. **Copy a File from HDFS to Local Filesystem**

   Download the file sample.txt from HDFS to your local system.

```
hdfs dfs -get /mydir/sample.txt /path/to/local/destination
```

Program: Cut, Copy, and Paste Between HDFS and Local File System

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import java.io.IOException;

public class HDFSFileOperations {

    public static void main(String[] args) {
        // HDFS and local paths
        String hdfsSourcePath = "/user/hadoop/source"; // HDFS source
        String hdfsDestinationPath = "/user/hadoop/destination"; // HDFS
destination
        String localPath = "C:/localdata"; // Local filesystem path

        try {
            // Create Hadoop configuration and FileSystem object
            Configuration conf = new Configuration();
            FileSystem hdfs = FileSystem.get(conf);

            // Copy file/directory from HDFS to Local
            System.out.println("Copying from HDFS to Local...");
            hdfs.copyToLocalFile(new Path(hdfsSourcePath), new Path(localPath));
            System.out.println("File/Directory copied to: " + localPath);

            // Copy file/directory from Local to HDFS
            System.out.println("Copying from Local to HDFS...");
            hdfs.copyFromLocalFile(new Path(localPath), new
Path(hdfsDestinationPath));
            System.out.println("File/Directory copied to HDFS: " +
hdfsDestinationPath);

            // Move file/directory within HDFS
            System.out.println("Moving within HDFS...");
            hdfs.rename(new Path(hdfsSourcePath), new
```

```
Path(hdfsDestinationPath));
            System.out.println("File/Directory moved to: " +
hdfsDestinationPath);

            // Close FileSystem object
            hdfs.close();

        } catch (IOException e) {
            System.err.println("Exception caught: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Explanation:

1. **Initialization:**
    a. A Configuration object is created to load Hadoop configurations.
    b. The FileSystem object provides access to the HDFS.
2. **Copy from HDFS to Local:**
    a. The copyToLocalFile method copies a file or directory from HDFS to the local filesystem.
3. **Copy from Local to HDFS:**
    a. The copyFromLocalFile method copies a file or directory from the local filesystem to HDFS.
4. **Move (Cut and Paste) within HDFS:**
    a. The rename method is used to move a file or directory from one location to another within HDFS.
5. **Error Handling:**
    a. The IOException is caught to handle potential file or path-related errors.
6. **Paths:**
    a. Replace hdfsSourcePath, hdfsDestinationPath, and localPath with actual paths as per your setup.

Program: Get File Status in HDFS

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.Path;

import java.io.IOException;

public class HDFSFileStatus {

    public static void main(String[] args) {
        // Specify the HDFS file path
        String hdfsFilePath = "/user/hadoop/sample.txt"; // Replace with your
file path

        try {
            // Create Hadoop Configuration
            Configuration conf = new Configuration();

            // Specify the NameNode URI (Optional, depending on your setup)
            conf.set("fs.defaultFS", "hdfs://localhost:9000");

            // Get the FileSystem object
            FileSystem hdfs = FileSystem.get(conf);

            // Get the status of the file
            Path filePath = new Path(hdfsFilePath);
            if (hdfs.exists(filePath)) {
                FileStatus fileStatus = hdfs.getFileStatus(filePath);

                // Print file status details
                System.out.println("File Path: " + fileStatus.getPath());
                System.out.println("Is Directory: " + fileStatus.isDirectory());
                System.out.println("Size: " + fileStatus.getLen() + " bytes");
                System.out.println("Owner: " + fileStatus.getOwner());
                System.out.println("Group: " + fileStatus.getGroup());
                System.out.println("Permissions: " +
fileStatus.getPermission());
                System.out.println("Last Modified: " +
```

```
fileStatus.getModificationTime());
                System.out.println("Replication Factor: " +
fileStatus.getReplication());
                System.out.println("Block Size: " + fileStatus.getBlockSize() +
" bytes");
            } else {
                System.out.println("The file does not exist in HDFS.");
            }

            // Close the FileSystem object
            hdfs.close();

        } catch (IOException e) {
            System.err.println("Exception occurred while getting file status: "
+ e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Explanation:

1. **Configuration Setup:**

   A Configuration object is initialized to load Hadoop configurations, including the fs.defaultFS parameter pointing to the HDFS NameNode.

2. **FileSystem Object:**

   The FileSystem object is used to interact with HDFS.

3. **Check File Existence:**

   The exists() method checks if the file exists in HDFS before attempting to retrieve its status.

4. **Retrieve File Status:**

   The getFileStatus() method fetches metadata about the file, which includes:

   1. File path
   2. Whether it's a file or directory

3. File size
4. Owner, group, and permissions
5. Last modified timestamp
6. Replication factor and block size

5. **Output:**

The program prints the file's metadata to the console.

6. **Error Handling:**

The program gracefully handles IOException to manage issues like file not found or connectivity problems.

Program: List Files in HDFS Directory

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.Path;

import java.io.IOException;

public class HDFSListFiles {

    public static void main(String[] args) {
        // Specify the HDFS directory path
        String hdfsDirectoryPath = "/user/hadoop"; // Replace with your HDFS
directory path

        try {
            // Create Hadoop Configuration
            Configuration conf = new Configuration();

            // Specify the NameNode URI (Optional, depending on your setup)
            conf.set("fs.defaultFS", "hdfs://localhost:9000");

            // Get the FileSystem object
            FileSystem hdfs = FileSystem.get(conf);

            // Get the list of files and directories
            Path directoryPath = new Path(hdfsDirectoryPath);
            if (hdfs.exists(directoryPath)) {
                FileStatus[] fileStatuses = hdfs.listStatus(directoryPath);

                System.out.println("Contents of directory: " +
hdfsDirectoryPath);
                for (FileStatus status : fileStatuses) {
                    System.out.println("----------------------------------------
---------");
                    System.out.println("Name: " + status.getPath().getName());
                    System.out.println("Path: " + status.getPath());
                    System.out.println("Is Directory: " + status.isDirectory());
                    System.out.println("Size: " + status.getLen() + " bytes");
```

```
                    System.out.println("Owner: " + status.getOwner());
                    System.out.println("Group: " + status.getGroup());
                    System.out.println("Permissions: " +
status.getPermission());
                    System.out.println("Replication Factor: " +
status.getReplication());
                    System.out.println("Block Size: " + status.getBlockSize() +
" bytes");
                }
            } else {
                System.out.println("The specified directory does not exist in
HDFS.");
            }

            // Close the FileSystem object
            hdfs.close();

        } catch (IOException e) {
            System.err.println("Exception occurred while listing files: " +
e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Explanation:

1. **Configuration Setup:** The Configuration object loads the Hadoop configuration and specifies the HDFS NameNode URI using fs.defaultFS.
2. **FileSystem Object:** The FileSystem object provides methods to interact with HDFS, such as listStatus().
3. **Check Directory Existence:** The exists() method ensures the specified directory exists in HDFS before attempting to list its contents.
4. **List Files and Directories:** The listStatus() method retrieves metadata for all files and directories in the specified HDFS directory.
5. **Output Metadata:**For each file or directory, the program prints its:

a. Name
b. Path
c. Type (file or directory)
d. Size
e. Owner, group, and permissions
f. Replication factor

Program: Implement MapReduce (Word Count Program)

Modified Program Description

1. **Main Function:** Accepts an argument to allow the user to assign the number of reducers.
2. **Mapper:** Changes functionality from WordCount to CharacterCount, ignoring spaces.
3. **Reducer:** Outputs characters that occur 20 or more times.
4. **Part II: Sorting:** Implements another MapReduce job to sort the output based on frequency.

Mapper for Character Count

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class CharacterCountMapper extends Mapper<Object, Text, Text,
IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text character = new Text();

    @Override
    protected void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString().replaceAll(" ", ""); // Remove spaces
        for (char c : line.toCharArray()) {
            character.set(String.valueOf(c)); // Convert character to String
            context.write(character, one);
        }
    }
}
```

Reducer for Filtering Character Count

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class CharacterCountReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        if (sum >= 20) { // Output characters with count >= 20
            context.write(key, new IntWritable(sum));
        }
    }
}
```

Driver Code for Part I

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class CharacterCount {

    public static void main(String[] args) throws Exception {
        if (args.length < 3) {
            System.err.println("Usage: CharacterCount <input path> <output
path> <num reducers>");
            System.exit(-1);
        }
```

```
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Character Count");

        job.setJarByClass(CharacterCount.class);
        job.setMapperClass(CharacterCountMapper.class);
        job.setReducerClass(CharacterCountReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Set number of reducers
        job.setNumReduceTasks(Integer.parseInt(args[2]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Sorting Mapper for Part II

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class SortingMapper extends Mapper<Object, Text, IntWritable, Text> {

    private IntWritable frequency = new IntWritable();
    private Text character = new Text();

    @Override
```

```
    protected void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        String[] fields = value.toString().split("\t");
        if (fields.length == 2) {
            character.set(fields[0]);
            frequency.set(Integer.parseInt(fields[1]));
            context.write(frequency, character);
        }
    }
}
```

Sorting Reducer for Part II

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class SortingReducer extends Reducer<IntWritable, Text, Text,
IntWritable> {

    @Override
    protected void reduce(IntWritable key, Iterable<Text> values, Context
context) throws IOException, InterruptedException {
        for (Text val : values) {
            context.write(val, key);
        }
    }
}
```

Driver Code for Part II

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
```

```java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class CharacterCountSorting {

    public static void main(String[] args) throws Exception {
        if (args.length < 2) {
            System.err.println("Usage: CharacterCountSorting <input path>
<output path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Character Count Sorting");

        job.setJarByClass(CharacterCountSorting.class);
        job.setMapperClass(SortingMapper.class);
        job.setReducerClass(SortingReducer.class);

        // Mapper output classes
        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(Text.class);

        // Final output classes
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Stack, Queue, and Linked List:

1. Stack Implementation

A Stack operates on a Last-In-First-Out (LIFO) principle.

```
import java.util.Stack;

public class StackExample {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();

        // Push elements onto the stack
        stack.push(10);
        stack.push(20);
        stack.push(30);
        System.out.println("Stack after push operations: " + stack);

        // Peek the top element
        System.out.println("Top element: " + stack.peek());

        // Pop elements from the stack
        System.out.println("Popped element: " + stack.pop());
        System.out.println("Stack after pop operation: " + stack);

        // Check if stack is empty
        System.out.println("Is stack empty? " + stack.isEmpty());
    }
}
```

Sample Output:

```
Stack after push operations: [10, 20, 30]
Top element: 30
Popped element: 30
Stack after pop operation: [10, 20]
Is stack empty? false
```

2. Queue Implementation

A Queue operates on a First-In-First-Out (FIFO) principle.

```
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();

        // Add elements to the queue
        queue.add(10);
        queue.add(20);
        queue.add(30);
        System.out.println("Queue after add operations: " + queue);

        // Peek the front element
        System.out.println("Front element: " + queue.peek());

        // Remove elements from the queue
        System.out.println("Removed element: " + queue.remove());
        System.out.println("Queue after remove operation: " + queue);

        // Check if queue is empty
        System.out.println("Is queue empty? " + queue.isEmpty());
    }
}
```

Sample Output:

```
Queue after add operations: [10, 20, 30]
Front element: 10
Removed element: 10
Queue after remove operation: [20, 30]
Is queue empty? false
```

3. Linked List Implementation

A Linked List allows dynamic memory allocation and consists of nodes connected by references.

Singly Linked List

```
class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class SinglyLinkedList {
    Node head;

    // Add a new node at the end
    public void add(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
        }
    }

    // Display the list
    public void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
```

```
        System.out.println("null");
    }

    // Delete the first node
    public void deleteFirst() {
        if (head != null) {
            head = head.next;
        }
    }

    public static void main(String[] args) {
        SinglyLinkedList list = new SinglyLinkedList();
        list.add(10);
        list.add(20);
        list.add(30);

        System.out.println("Linked List after additions:");
        list.display();

        list.deleteFirst();
        System.out.println("Linked List after deleting the first node:");
        list.display();
    }
}
```

Sample Output:

```
Linked List after additions:
10 -> 20 -> 30 -> null
Linked List after deleting the first node:
20 -> 30 -> null
```

Using Java's Built-in LinkedList

```
import java.util.LinkedList;
```

```
public class BuiltInLinkedListExample {
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<>();

        // Add elements to the list
        list.add(10);
        list.add(20);
        list.add(30);
        System.out.println("Linked List: " + list);

        // Remove the first element
        list.removeFirst();
        System.out.println("After removing the first element: " + list);

        // Get the first element
        System.out.println("First element: " + list.getFirst());
    }
}
```

Sample Output:

```
Linked List: [10, 20, 30]
After removing the first element: [20, 30]
First element: 20
```

Program: Wrapper Class in java

Wrapper classes in Java are used to wrap primitive data types into objects. Each primitive data type has a corresponding wrapper class:

Primitive Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

```
public class WrapperClassExample {
    public static void main(String[] args) {
        // 1. Boxing (Primitive to Wrapper Object)
        Integer intObj = Integer.valueOf(10); // Explicit Boxing
        Integer autoBoxed = 20;               // Autoboxing
        System.out.println("Boxed Integer: " + intObj);
        System.out.println("Autoboxed Integer: " + autoBoxed);

        // 2. Unboxing (Wrapper Object to Primitive)
        int intValue = intObj.intValue();     // Explicit Unboxing
        int autoUnboxed = autoBoxed;          // Autounboxing
        System.out.println("Unboxed Integer: " + intValue);
        System.out.println("Auto-unboxed Integer: " + autoUnboxed);

        // 3. Utility Methods in Wrapper Classes
        // Parsing a string to a primitive
```

```
        String strNum = "50";
        int parsedInt = Integer.parseInt(strNum);
        System.out.println("Parsed Integer: " + parsedInt);

        // Getting the maximum value for Integer
        System.out.println("Max Value of Integer: " + Integer.MAX_VALUE);

        // Converting Primitive to String
        String floatToString = Float.toString(5.75f);
        System.out.println("Float to String: " + floatToString);

        // 4. Checking Number Properties
        System.out.println("Is 10 a NaN? " + Float.isNaN(10.0f)); // Check if
it's NaN
        System.out.println("Is 0/0 a NaN? " + Float.isNaN(0.0f / 0.0f)); //
Check if it's NaN

        // 5. Character Wrapper Class
        char ch = 'A';
        Character charObj = ch; // Autoboxing
        System.out.println("Character Object: " + charObj);
        System.out.println("Is Letter? " + Character.isLetter(charObj));
        System.out.println("Is Digit? " + Character.isDigit(charObj));
        System.out.println("Lowercase: " + Character.toLowerCase(charObj));

        // 6. Boolean Wrapper Class
        Boolean boolObj = Boolean.valueOf(true);
        System.out.println("Boolean Object: " + boolObj);
        System.out.println("Boolean as Primitive: " +
boolObj.booleanValue());
    }
}
```

Explanation

1. Boxing and Autoboxing:
    - Integer.valueOf(10) explicitly wraps the primitive 10 into an Integer object.
    - Assigning 20 to an Integer variable automatically boxes it.
2. Unboxing and Autounboxing:

- .intValue() explicitly converts the Integer object back to a primitive int.
- Assigning an Integer object to a primitive int variable automatically unboxes it.

3. **Utility Methods:**
   - Integer.parseInt("50") converts a String into a primitive int.
   - Integer.MAX_VALUE gives the maximum value an int can hold.
4. **Character Class:**
   - Methods like Character.isLetter(char) and Character.toLowerCase(char) help in character processing.
5. **Boolean Class:**
   - Boolean.valueOf(true) wraps a primitive boolean into a Boolean object.

Output

```
Boxed Integer: 10
Autoboxed Integer: 20
Unboxed Integer: 10
Auto-unboxed Integer: 20
Parsed Integer: 50
Max Value of Integer: 2147483647
Float to String: 5.75
Is 10 a NaN? false
Is 0/0 a NaN? true
Character Object: A
Is Letter? true
Is Digit? false
Lowercase: a
Boolean Object: true
Boolean as Primitive: true
```

Program: Serialization in java

Serialization in Java is a mechanism to convert the state of an object into a byte stream, enabling the object to be saved to a file or transmitted over a network. Deserialization is the reverse process, where the byte stream is converted back into an object.

```
import java.io.*;

class Person implements Serializable {
    private static final long serialVersionUID = 1L;

    private String name;
    private int age;
    private transient String password; // Will not be serialized

    public Person(String name, int age, String password) {
        this.name = name;
        this.age = age;
        this.password = password;
    }

    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + ", password='" +
password + "'}";
    }
}

public class SerializationExample {
    public static void main(String[] args) {
        // Create an object of Person
        Person person = new Person("John Doe", 30, "secret123");

        // Serialize the object
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("person.ser"))) {
            oos.writeObject(person);
            System.out.println("Object has been serialized: " + person);
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }

        // Deserialize the object
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("person.ser"))) {
            Person deserializedPerson = (Person) ois.readObject();
            System.out.println("Object has been deserialized: " +
deserializedPerson);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Explanation

1. **Serialization:**
    a. The `ObjectOutputStream` writes the `Person` object to a file (`person.ser`).
    b. The `password` field is marked as `transient`, so its value will not be saved.
2. **Deserialization:**
    a. The `ObjectInputStream` reads the object back from the file and reconstructs it.
    b. Since `password` is `transient`, its value will be `null` after deserialization.

Output

```
Object has been serialized: Person{name='John Doe', age=30,
password='secret123'}
Object has been deserialized: Person{name='John Doe', age=30,
password='null'}
```