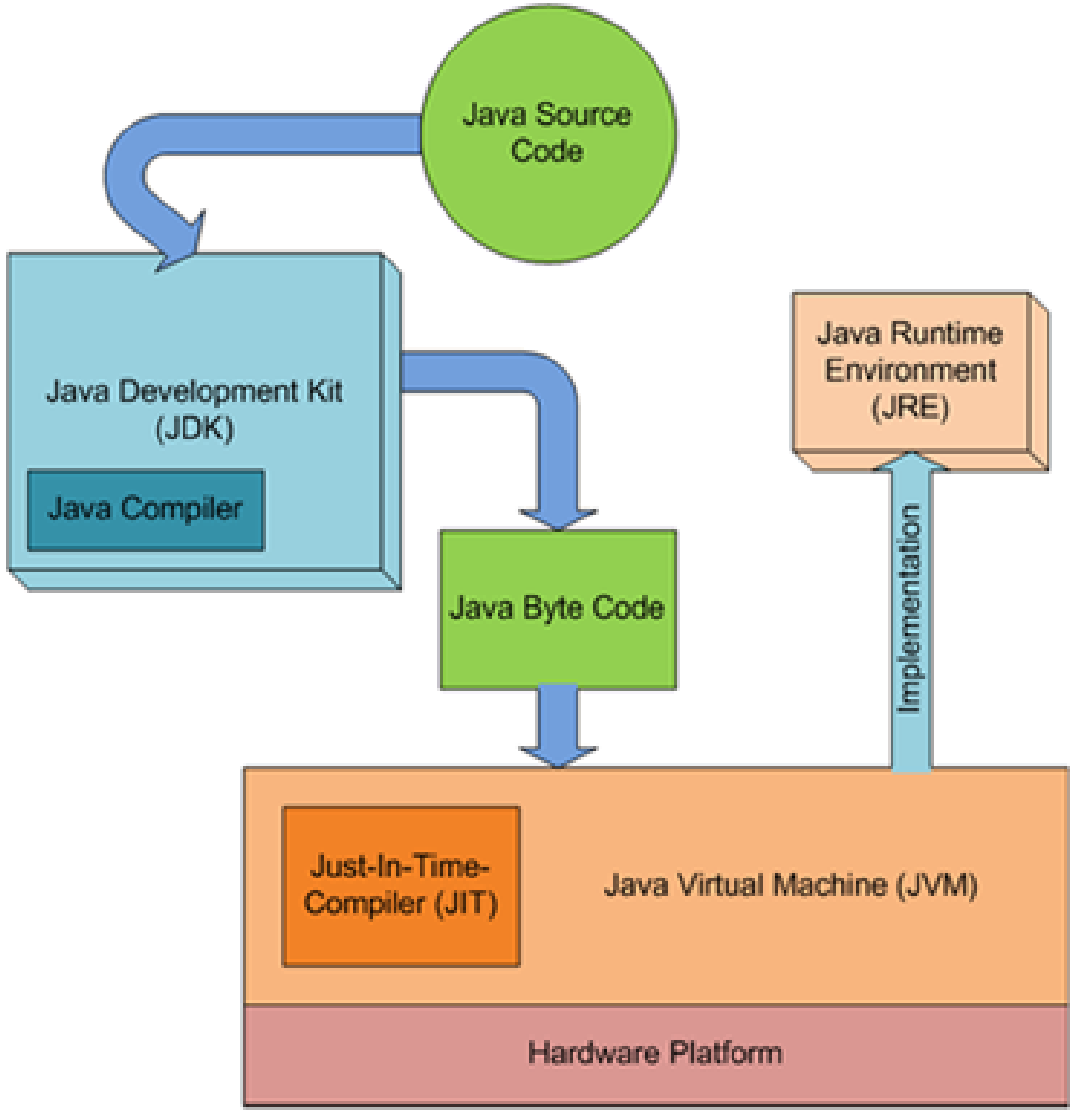
JAVA PROGRAMMING LANGUAGE

Basic Introduction

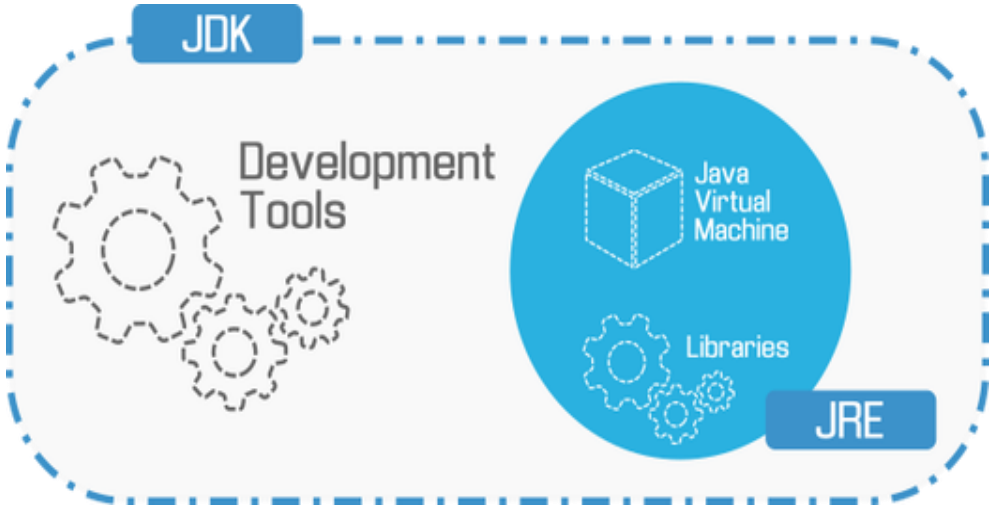
INTRODUCTION

- JAVA is designed to be a platform-independent language, which means that code written in JAVA can run on any device or operating system that has a JAVA Virtual Machine (JVM) installed.



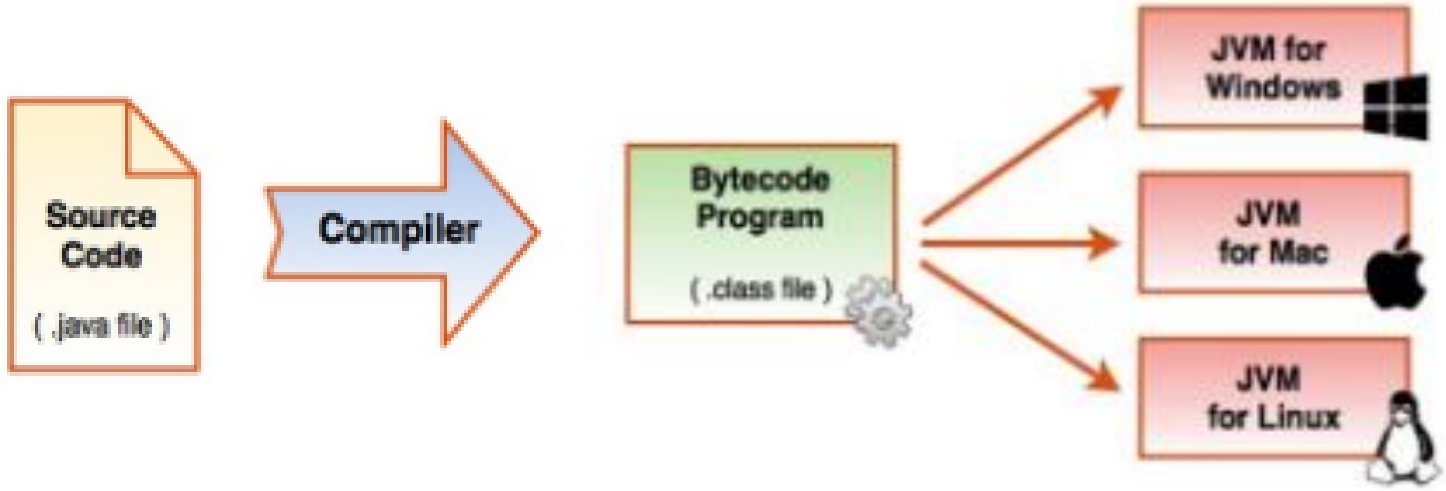
- JDK stands for Java Development Kit, which is a software development kit used by developers to create JAVA applications. The JDK includes a set of programming tools, such as the JAVA compiler, debugger, and documentation, which are essential for developing JAVA applications.

- JVM stands for Java Virtual Machine, which is the software that executes JAVA code. When a JAVA application is executed, the code is compiled by the JAVA compiler and converted into bytecode, which can be executed by the JVM. The JVM is responsible for interpreting the bytecode and executing the application on the underlying operating system.



STEPS TO CREATE AND RUN JAVA FILE

- Install the Java Development Kit (JDK) on your computer. You can download the latest version of the JDK from the Oracle website.
- Open a text editor like Notepad, Sublime Text, or Visual Studio Code.
- Create a new file and save it with a .java extension. For example, you can name the file "Main.java"



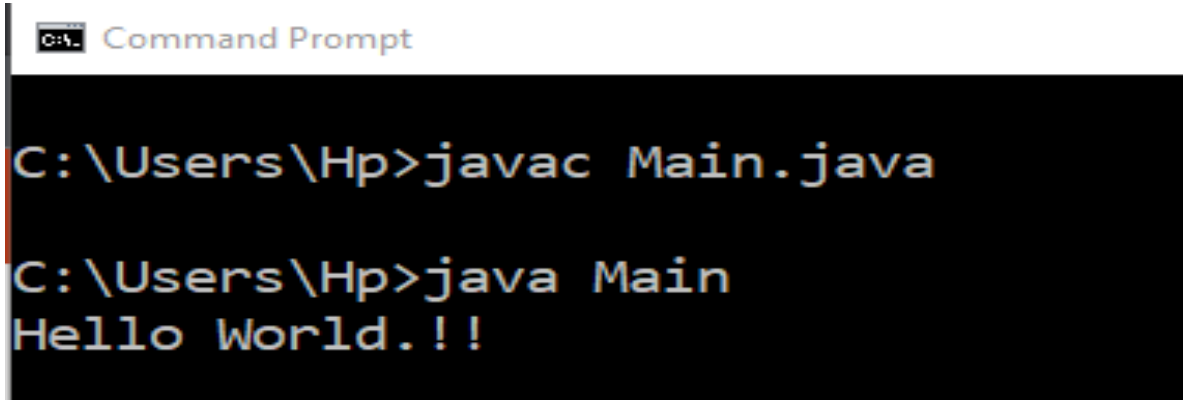
- Example:

```
public class Main{
    public static void main(String[] args){
        System.out.println("Hello World.!!");
    }
}
```

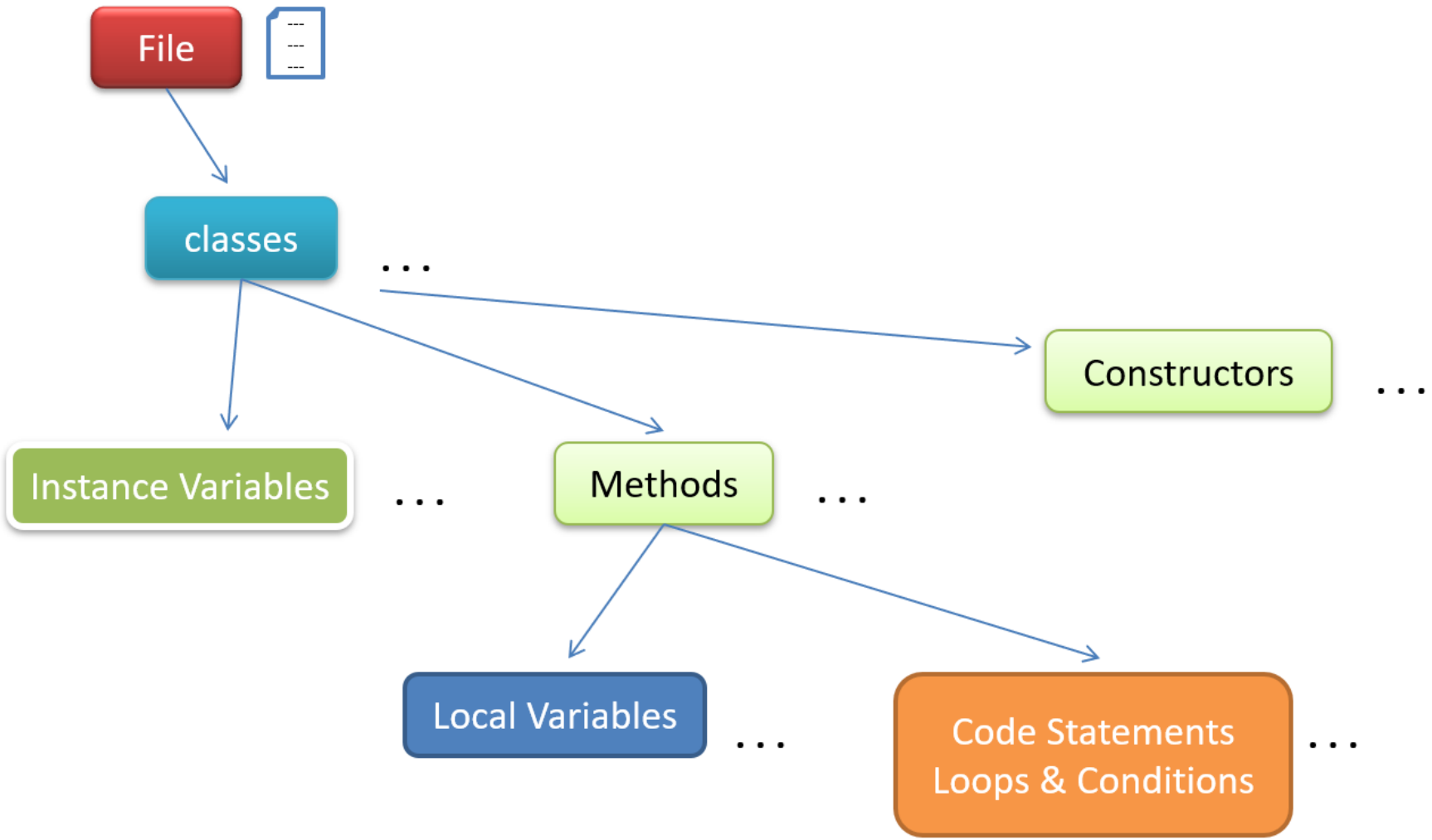
Main.java

Main.class

Commands:



Java Code Organization:



PRIMITIVE DATA TYPES IN JAVA

- **byte**: This data type is used to store integer values between -128 and 127.
- **short**: This data type is used to store integer values between -32,768 and 32,767.
- **int**: This data type is used to store integer values between -2,147,483,648 and 2,147,483,647.
- **long**: This data type is used to store integer values between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807.
- **float**: This data type is used to store floating-point values with single precision.
- **double**: This data type is used to store floating-point values with double precision.
- **boolean**: This data type is used to store values that are either **true** or **false**.
- **char**: This data type is used to store a single character or letter.

- Example:

```
public class Main{
    public static void main(String[] args){
        byte b = 42;
        Boolean flag = false;
        Int  x =  324584;
        System.out.println(b);          //output: 42
        System.out.println(flag);       // output: false
        System.out.println(x);           // output: 324584
    }
}
```

DIFFERENT WAYS TO READ INPUT FROM CONSOLE

- Using Scanner class:

The Scanner class provides a convenient way to read input from the console. You can use the **`Scanner`** class to read input from the console by creating a new **`Scanner`** object and then using the **`next`** or **`nextLine`** method to read the input.

- Using BufferedReader class:

Another way to read input from the console is to use the **`BufferedReader`** class. This class provides a method called **`readLine()`** that reads a line of text from the console.

- Using Console class:

If you are using Java 6 or later, you can use the **`Console`** class to read input from the console. This class provides methods like **`readLine()`** and **`readPassword()`** to read input.

- Examples:

```
import java.util.Scanner;
public class ReadInput {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = scanner.nextLine();
        System.out.println("You entered: " + str);
        scanner.close();
    }
}
```

- Examples:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class ReadInput {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter a string: ");
        String str = reader.readLine();
        System.out.println("You entered: " + str);
        reader.close();
    }
}
```

FILE READING AND WRITING

- Using FileReader and FileWriter:

The FileReader and FileWriter classes provide a convenient way to read and write character data from/to a file. You can use these classes to read/write data one character at a time.

- Using BufferedReader and BufferedWriter:

The BufferedReader and BufferedWriter classes provide a more efficient way to read and write data from/to a file. You can use these classes to read/write data in chunks.

- Using FileInputStream and FileOutputStream:

The FileInputStream and FileOutputStream classes provide a way to read and write binary data from/to a file. You can use these classes to read/write data in bytes.

- Example: File Reader and File Writer

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class FileReadWrite {
    public static void main(String[] args) throws IOException {
        FileReader reader = null;
        FileWriter writer = null;
        try {
            reader = new FileReader("input.txt");
            writer = new FileWriter("output.txt");
            int ch;
            while ((ch = reader.read()) != -1) {
                writer.write(ch);
            }
        }
```

```
 finally {
            if (reader != null) {
                reader.close();
            }
            if (writer != null) {
                writer.close();
            }
        }
    }
}
```

Example: Buffered Reader and Buffered Writer

```
import java.io.*;
public class FileReadWrite {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = null;
        BufferedWriter writer = null;
        try {
            reader = new BufferedReader(new FileReader("input.txt"));
            writer = new BufferedWriter(new FileWriter("output.txt"));
            String line;
            while ((line = reader.readLine()) != null) {
                writer.write(line);
                writer.newLine();
            }
        }
```

```
 finally {
            if (reader != null) {
                reader.close();
            }
            if (writer != null) {
                writer.close();
            }
        }
    }
}
```

LINKEDLIST IMPLEMENTATION

1. Define a class to represent the nodes of the linked list. Each node should have a data field to store the data and a reference field to point to the next node in the list.

- Example:

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

2) Create an instance of the **Node** class for each element of the linked list

```
Node head = new Node(1);
```

Example: create a Linkedlist of size 10 containing numbers from 1 to 10

```
class LinkedList{
    public static void main(String[] args){
        Node head = null,tail=null;
        //create linked list of size 10 which contains numbers from 1 to 10
        for(int i=1;i<=10;i++){
            Node temp = new Node(i);
            if(head==null){
                head=tail=temp;
            }else{
                tail.next = temp;
                tail = temp;
            }
        }
```

```
// print the linkedlist
        print(head);
//output: 1 2 3 4 5 6 7 8 9 10
    }
    static void print(Node head){
        while(head!=null){
            System.out.print(head.data+" ");
            head = head.next;
        }
        System.out.println();
    }
}
```

- Just like LinkedList we can also implement other data structure.
  - Stack
  - Queue
  - Deque

Java 10 API Documentation:

https://docs.oracle.com/javase/10/docs/api/index.html?overview-summary.html

Here you can check how different utility classes are implemented, like ArrayList, Stack, Queue(Interface), PriorityQueue, Deque etc.

ARRAY

- One-dimensional array: A one-dimensional array is the simplest type of array, where the data elements are stored in a single row. It is also known as a vector or a list.
  - Example: dataType[] arrayName = new dataType[arraySize];

    int[] numbers = new int[5];

- Multidimensional array: A multidimensional array is an array that has more than one row and column. It is also known as a matrix or a table. In Java, you can create two-dimensional arrays, three-dimensional arrays, and so on.

  Example: 2 D Array:

  dataType[][] arrayName = new dataType[arraySize][arraySize];

  int[][] numbers = new int[5][5];

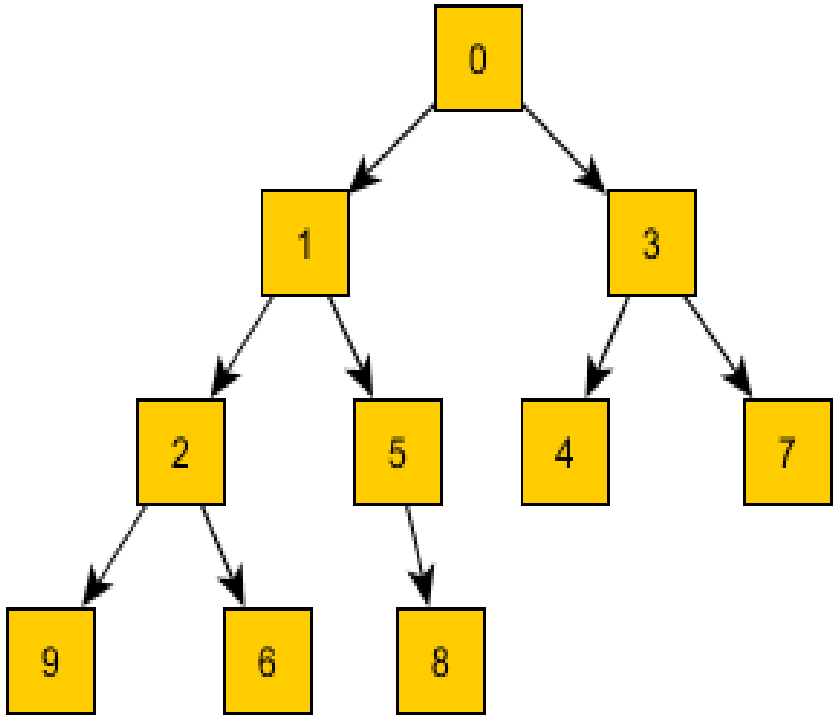
- Example

```
public class OneDArray{
    public static void main(String[] args){
        int[] arr = new int[5];
        for(int i=0;i<5;i++){
            arr[i] = i+1;
        }
        for(int i=0;i<5;i++){
            System.out.print(arr[i]+" ");
        }
    }
}
```

HEAP (MIN OR MAX)

- A heap is a data structure that is used to maintain a collection of elements with a specific ordering. A heap is a binary tree where each node has a value that is greater than or equal to (for a max heap) or less than or equal to (for a min heap) the values of its child nodes.

- A heap is commonly used to implement priority queues, where the elements are sorted by their priority.



Min Heap

```
class MinHeap{
    static void heapify(int[] arr,int i,int n){
        int idx = i,l = 2*i+1,r = l+1;
        if(l<n && arr[l]<arr[idx]){
            idx = l;
        }
      if(r<n && arr[r]<arr[idx]){
            idx = r;
        }
        if(i!=idx){
            int t = arr[idx];
            arr[idx] = arr[i];
            arr[i] = t;
            heapify(arr,idx,n);
        }
    }
```

```
    static void buildHeap(int[] arr,int n){
        for(int i=n/2-1;i>=0;i--){
            heapify(arr,i,n);
        }
    }
    public static void main(String[] args){
        int[] arr = {10,4,15,1,5,8,2};
        int n = arr.length;
        buildHeap(arr,n);
        for(int i=0;i<n;i++){
            System.out.print(arr[i]+" ");
        }
    }
}
```

THANK YOU