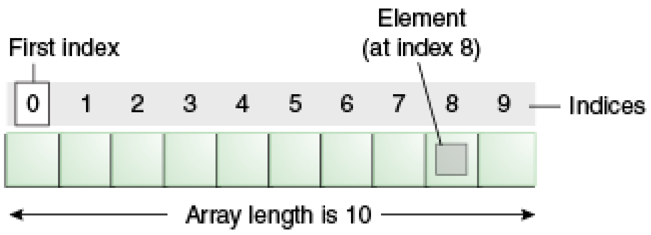
Write a java program to demonstrate the array concepts in java.
Or
What is an array? Explain different type of arrays and discuss its advantages.
Or
Write a java program to implement the different array declarations.

Java array is an object which contains elements of a similar data type. Additionally, the elements of an array are stored in a contiguous memory location.

It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on the 1st Index, and so on.



TYPES OF ARRAYS IN JAVA

There are two types of arrays.

- One Dimensional Array
- Two-Dimensional Array
- Variable sized araay

One Dimensional Array in Java

Syntax to Declare an Array in Java
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];
Instantiation of an Array in Java
arrayRefVar=new datatype[size];
Example:
int intArray[]; //declaring array
intArray = new int[20]; // allocating memory to array
or
int intArray[] = new int[20]; // combining both statements in one

Two-Dimensional Array in Java

In such cases, data is stored in a row and column-based index (also known as matrix form).

Syntax

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][];
```

Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3]; //3 row and 3 column
```

Variable sized Array in Java

A variable size array is an array that can have different lengths for each row.

Syntax:

```
dataType[][] arrayName = new dataType[rowSize][];  // declares number of rows
arrayName[rowIndex] = new dataType[columnSize];  // To define a number column
arrayName[rowIndex][columnIndex] = value;                  // To Assign a value
```

ADVANTAGES

- **Code Optimization**: It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access**: We can get any data located at an index position.

DISADVANTAGES

Size Limit: We can store only the fixed size of elements in the array. It does not grow its size at runtime. To solve this problem, a collection framework is used in Java which grows automatically.

PROGRAM

//Java Program to demonstrate the single dimension, multi dimension and variable sized array in Java

```
class ArrayDemo {
  public static void main (String args[]) {
    //declare and initialize a single dimensional array of integers
    int[] intArray = {10, 20, 30, 40};
    //declare and initialize a two dimensional array of doubles
    double[][] dblArray = {{1.1, 2.2}, {3.3, 4.4}, {5.5, 6.6}};

    //declare and initialize a variable size array of strings
    String[][] strArray = new String[2][];
    strArray[0] = new String[2]; //first row has 2 columns
    strArray[1] = new String[3]; //second row has 3 columns

    //assign values to each element of the variable size array
    strArray[0][0] = "Hello";
    strArray[0][1] = "World";
    strArray[1][0] = "Java";
    strArray[1][1] = "Arrays";
    strArray[1][2] = "Are";

    //print the elements of each array using nested for loops
    System.out.println("The elements of the intArray are: ");
    for(int i = 0; i < intArray.length; i++) {
      System.out.print(intArray[i] + " ");
    }
    System.out.println();
    System.out.println("The elements of the dblArray are: ");
    for(int i = 0; i < dblArray.length; i++) {
      for(int j = 0; j < dblArray[i].length; j++) {
        System.out.print(dblArray[i][j] + " ");
      }
      System.out.println();
    }
    System.out.println("The elements of the strArray are: ");
    for(int i = 0; i < strArray.length; i++) {
      for(int j = 0; j < strArray[i].length; j++) {
        System.out.print(strArray[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```

OUTPUT

```
The elements of the intArray are:
10 20 30 40
The elements of the dblArray are:
1.1 2.2
3.3 4.4
5.5 6.6
The elements of the strArray are:
Hello World
Java Arrays Are
```

Write a program in java that create and initialize four integer elements in an array and find the sum and average.

We need to store four integer numbers so single dimensional array is enough for this program.

One Dimensional Array in Java

Syntax to Declare an Array in Java

```
dataType[] arr; (or)
dataType arr[];
```

Instantiation of an Array in Java

```
arrayRefVar=new datatype[size];
```

PROGRAM

```
public class ArrayExample {
   public static void main(String[] args) {
      // Create an array of four integer elements
      int[] array = {10, 20, 30, 40};
      // Declare variables to store the sum and average
      int sum = 0;
      double average = 0.0;
     // Loop through the array and add each element to the sum
      for (int i = 0; i < array.length; i++) {
         sum += array[i];
      }
       // Calculate the average by dividing the sum by the array length
      average = (double) sum / array.length;
      // Print the sum and average
      System.out.println("The sum of the array elements is: " + sum);
      System.out.println("The average of the array elements is: " + average);
   }
}
```

Output:

```
The sum of the array elements is: 100
The average of the array elements is: 25.0
```

Create a java application to get N numbers of string and display them using an array.

```
import java.util.Scanner;
public class StringArrayExample {
  public static void main(String[] args) {
    // Create a scanner object to get user input
    Scanner sc = new Scanner(System.in);

    // Ask the user to enter the number of strings
    System.out.println("Enter the number of strings: ");
    int n = sc.nextInt();

    // Create an array of strings with size n
    String[] array = new String[n];

    // Ask the user to enter each string and store it in the array
    System.out.println("Enter " + n + " strings: ");
    for (int i = 0; i < n; i++) {
      array[i] = sc.next();
    }

    // Close the scanner object
    sc.close();

    // Display the array elements
    System.out.println("The array elements are: ");
    for (int i = 0; i < n; i++) {
      System.out.println(array[i]);
    }
  }
}
```

Output

```
Enter the number of strings:
3
Enter 3 strings:
CSE
IT
ECE
The array elements are:
CSE
IT
ECE
```

What is final keyword? Explain its importance in java with an example program.

Final keyword can be used along with variables, methods, and classes.

1. final variable
2. final methods
3. final class

1. Final variable

A final variable is a variable whose value cannot be changed at any time once assigned, it remains as a constant forever.

Example:

```
public class Travel
{
final int SPEED=60;
void increaseSpeed()
{
 SPEED=70;
   }
public static void main(String args[])
{
   Travel t=new Travel();
   t.increaseSpeed();
}
}
```

Output

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The final field Travel.SPEED cannot be assigned.
```

Explanation

The above code will give you Compile time error, as we are trying to change the value of a final variable 'SPEED'.

2.final method

When a method is declared as final, it is called as final method. A final method cannot be overridden.

Example

```
class Parent
{
        final void disp()
        {
                System.out.println("disp() method of parent class");
        }
}
class Child extends Parent
{
        void disp()
        {
                 System.out.println("disp() method of child class");
        }
}

class Demo
{
        public static void main(String args[])
        {
           Child c = new Child();
           c.disp();
        }
}
```

Output:

We will get the below error as we are overriding the disp() method of the Parent class.

```
Exception in thread "main" java.lang.VerifyError: class com.javainterviewpoint.Child overrides
final method disp.()

at java.lang.ClassLoader.defineClass1(Native Method)

at java.lang.ClassLoader.defineClass(Unknown Source)

at java.security.SecureClassLoader.defineClass(Unknown Source)
```

3.final class

When a class is declared as final, it cannot be extended. Final classes prevent Inheritance. If you try, it gives a compile time error.

Example

```
final class myFinalClass
```

```
{
   void myMethod()
   {
      System.out.println("We are in the final class we just created");
   }
}
class subClass extends myFinalClass
{
   void myMethod()
   {
      System.out.println("We are in the subclass");
   }
}
class MainClass
{
   public static void main(String arg[])
   {
      myFinalClass fc = new subClass();
      fc.myMethod();
   }
}
```

Output :

```
          error: cannot inherit from final myFinalClass
```

What is Inheritance? Discuss its uses and hierarchical abstractions?

Or

What is an inheritance? Explain the benefits of inheritance with an example.

Or

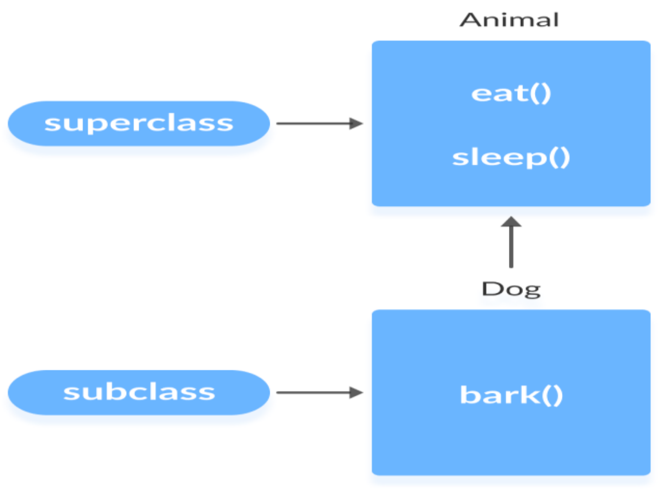
List different types of inheritance in java. Explain each of them in detail with an example program.

Or

Explain different types of inheritance in java using a suitable example.

- Inheritance can be defined as the process of acquiring all the properties and behaviors of one class to another.

- Acquiring the properties and behavior of child class from the parent class. Inheritance represents the **IS-A relationship**, also known as parent-child relationship



Advantages of Inheritance in Java

- For Method Overriding (so runtime polymorphism can be achieved)
- For Code Reusability

Syntax:

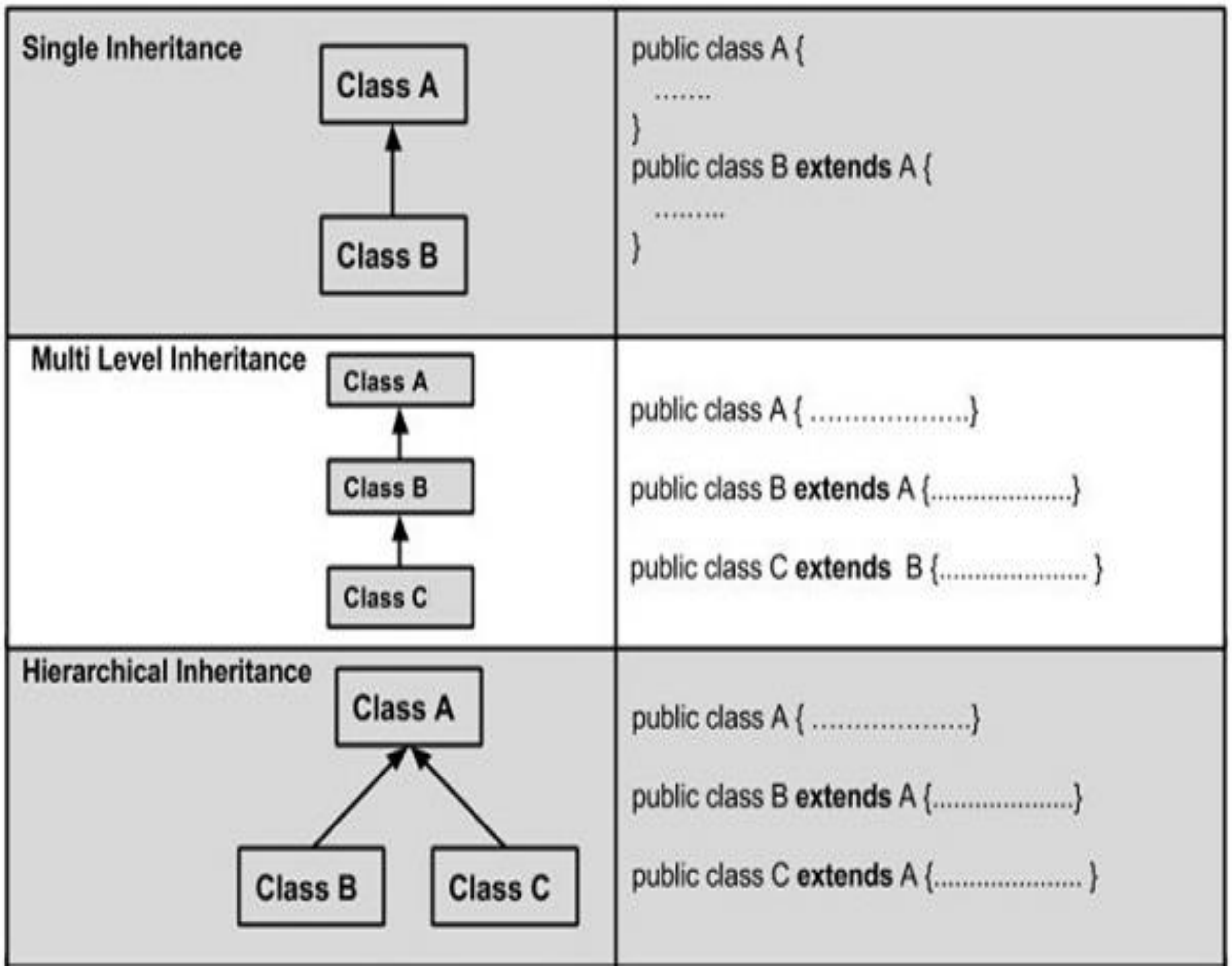
```
class subClass extends superClass
{
//methods and fields
}
```

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Types of Inheritance and Syntax

1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Hybrid inheritance – combination of more than one type of inheritance



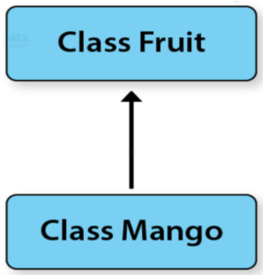
```
public class A {
  ........
}
public class B extends A {
  .........
}
```

```
public class A { ...................}
public class B extends A {...................}
public class C extends  B {.................... }
```

```
public class A { ...................}
public class B extends A {...................}
public class C extends A {.................... }
```

1.Single Inheritance

Process of extending single subclass from single super class



Example

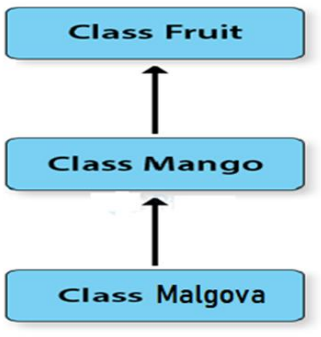
```
class Fruit
{
        String taste = "Sweet";
        void printTaste()
        {
                System.out.println(“Taste is: " + taste);
        }
}
class Mango extends Fruit
{
        String name = “Mango";
        void printName()
        {
                System.out.println("Name is: " + name);
        }
}
class SingleDemo
{
        public static void main(String args[])
        {
                Mango m1 = new Mango();
                m1.printTaste();
                m1.printName();
        }
}
```

Output:

```
        Taste is: Sweet

        Name is: Mango
```

2.Multilevel Inheritance

Process of extending subclass from another sub class



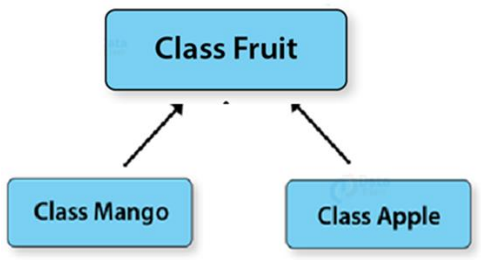
```
class Fruit

{

            String taste = “Sweet";

            void printTaste()

            {

                        System.out.println(“Taste is: " + taste);

            }

}

class Mango extends Fruit
{
            String name = “Mango";
            void printName()
            {
                        System.out.println("Name is: " + name);
            }
}
class Malgova extends Mango
{
            int weight = 2;
            void printWeight()
            {
                        System.out.println(“Weight is: " + weight);
            }
}
class MultilevelDemo
{
            public static void main(String args[])
            {
                        Malgova m1 = new Malgova();
                        m1.printTaste();
                        m1.printName();
                                    m1.printWeight();
            }
}
```

Output:

```
               Taste is: Sweet
               Name is: Mango
               Weight is: 2
```

3. Hierarchical Inheritance

Process of extending more than one subclasses from single super class.



```
class Fruit
{
        String taste = “Sweet";
        void printTaste()
        {
                System.out.println(“Taste is: " + taste);
        }
}
class Mango extends Fruit
{
        String name = “Mango";
        void printName()
        {
                System.out.println("Name is: " + name);
        }
}
class Apple extends Fruit
{
        String name = “Apple";
        void printName()
        {
                System.out.println("Name is: " + name);
        }
}
class HierarchicalDemo
{
        public static void main(String args[])
        {
                Mango m1 = new Mango();
                m1.printTaste();
                m1.printName();
                Apple a1 = new Apple();
                a1.printTaste();
                a1.printName();
        }
}
```

Output

```
    Taste is: Sweet
    Name is: Mango
    Taste is: Sweet
    Name is: Apple
```

Write a java program to illustrate reading input from the user in the command line environment (console) using Buffered Reader class and Scanner class.

There are two ways to read input from the user in the command line environment (console) in Java.

1. using Buffered Reader class and
2. using Scanner class.

Buffered Reader class

- It is a Java class that can read text from a character-input stream, buffering characters so as to provide for the efficient reading of characters.
- To use Buffered Reader class, we need to import java.io package and wrap the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader.
- Then we can use the readLine() method to read a line of text from the user.

Scanner class

- It is a Java class that can parse primitive types and strings using regular expressions, and also can read input from the user in the command line1.
- To use Scanner class, we need to import java.util package and create a Scanner object with System.in as the parameter.
- Then we can use various methods like nextInt(), nextFloat(), nextLine(), etc. to read input of specific types from the user.

Example

```
//Java Program to illustrate reading input from the user in the command line environment (console)
using Buffered Reader class and Scanner class
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

class InputDemo {
  public static void main(String args[]) throws IOException {
    //Using Buffered Reader class
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter your name using Buffered Reader: ");
    String name = reader.readLine();
    System.out.println("Hello, " + name + "!");

    //Using Scanner class
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter your age using Scanner: ");
    int age = scanner.nextInt();
    System.out.println("You are " + age + " years old.");
  }
}
```

Write a java program to illustrate reading input from the user in the command line environment (console) using Console class. List out advantages.

- Console class is a Java class that provides methods to access the character-based console device
- Console class can be used to read input from the user in the command line environment by using the following methods.
  1. readLine() – To read single line of text
  2. readPassword() – to read password without echoing the entered characters

The advantages of using Console class are:

1. Reading password without echoing the entered characters
2. Reading methods are synchronized.
3. Format string syntax can be used.

Drawback:

Does not work in non-interactive environment (such as in an IDE).

Program

```
//Java Program to illustrate reading input from the user in the command line environment (console)
using Console class

import java.io.Console;


class ConsoleDemo {
  public static void main(String args[]) {
    //Get the instance of Console class
    Console console = System.console();
    //Check if console device exists
    if(console != null) {
      //Read a string from the console
      System.out.println("Enter your name: ");
      String name = console.readLine();
      //Read a password from the console
      System.out.println("Enter your password: ");
      char[] password = console.readPassword();
      //Convert password to string
      String pass = String.valueOf(password);
      //Print the input
      System.out.println("Hello, " + name + "!");
      System.out.println("Your password is: " + pass);
    }
    else {
      System.out.println("No console device found.");
    }
  }
}
```

Output:

```
Enter your name: Akash
Enter your password: ***

Hello, Akash!

Your password is: 123    ( Entered password)
```

Write a java program to demonstrate various Scanner class methods to read primitive data.

Scanner class is a Java class that can parse primitive types and strings using regular expressions, and can read input from the user in the command line. To use Scanner class, we need to import java.util package and create a Scanner object with System.in as the parameter.

Scanner class provides various methods to read and parse different primitive data types from the user input, such as:

1. nextInt() - reads the next token of the input as an int value1.
2. nextLong() - reads the next token of the input as a long value1.
3. nextDouble() - reads the next token of the input as a double value1.
4. nextFloat() - reads the next token of the input as a float value1.
5. nextBoolean() - reads the next token of the input as a boolean value1.
6. nextByte() - reads the next token of the input as a byte value1.
7. nextShort() - reads the next token of the input as a short value1.
8. next() - reads the next token of the input as a String value1.
9. nextLine() - reads the rest of the current line of the input as a String value

Advantages of Scanner class:

- Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
- Regular expressions can be used to find tokens.

Program

```
//Java Program to demonstrate various Scanner class methods to read primitive data
import java.util.Scanner;
class ScannerDemo {
  public static void main(String args[]) {
```

```
 //Create a Scanner object
 Scanner scanner = new Scanner(System.in);
 //Read an int value
 System.out.println("Enter an int value: ");
 int i = scanner.nextInt();
 //Read a long value
 System.out.println("Enter a long value: ");
 long l = scanner.nextLong();
 //Read a double value
 System.out.println("Enter a double value: ");
 double d = scanner.nextDouble();
 //Read a float value
 System.out.println("Enter a float value: ");
 float f = scanner.nextFloat();
 //Read a boolean value
 System.out.println("Enter a boolean value: ");
 boolean b = scanner.nextBoolean();
 //Read a byte value
 System.out.println("Enter a byte value: ");
 byte by = scanner.nextByte();
 //Read a short value
 System.out.println("Enter a short value: ");
 short s = scanner.nextShort();
 //Read a String value
 System.out.println("Enter a String value: ");
 String str = scanner.next();
 //Read a line of text
 System.out.println("Enter a line of text: ");
 scanner.nextLine(); //consume the newline character left by scanner.next()
 String line = scanner.nextLine();

 //Print the input values
 System.out.println("You entered int: " + i);
 System.out.println("You entered long: " + l);
 System.out.println("You entered double: " + d);
 System.out.println("You entered float: " + f);
 System.out.println("You entered boolean: " + b);
 System.out.println("You entered byte: " + by);
 System.out.println("You entered short: " + s);
 System.out.println("You entered String: " + str);
 System.out.println("You entered line: " + line);
}
```

```
}
```
Input/Output

```
Enter an int value:  5
Enter a long value:  22225
Enter a double value: 20.5
Enter a float value: 10.1
Enter a boolean value: true
Enter a byte value: 2
Enter a short value: 155
Enter a String value: CSE
Enter a line of text:  this is java program
You entered int: 5
You entered long: 22225
You entered double: 20.5
You entered float: 10.1
You entered boolean: true
You entered byte: 2
You entered short: 155
You entered String: CSE
You entered line: this is java program
```

Explain standard I/O streams which is automatically created with the console with example java program.

- A stream is a flowing sequence of data that a user can either read from or write to
- A stream is attached to a data source or a data destination, such as a file, a device, a network socket.

Java provides three standard I/O streams that are automatically created with the console:

1. **System.in –**

   This is the standard input stream that reads data from the keyboard. It is an object of type

   InputStream.

2. **System.out** - this is the standard output stream that writes data to the console. It is an object of type PrintStream.

3. **System.err** - this is the standard error stream that writes error messages to the console. It is also an object of type PrintStream3.

- To use these streams, we can use various methods and classes provided by Java I/O package.

- For example, we can use Scanner class to read input from System.in, and we can use println() method to write output to System.out or System.err.

```
//Java Program to demonstrate standard I/O streams
import java.util.Scanner;

class StandardIOStreamsDemo {
  public static void main(String args[]) throws Exception {
    //Create a Scanner object to read input from System.in
    Scanner scanner = new Scanner(System.in);
    //Read a string from the user
    System.out.print("Enter your name: ");
    String name = scanner.nextLine();
    //Read an int value from the user
    System.out.print("Enter your age: ");
    int age = scanner.nextInt();
    // read int directly from console
    System.out.print("Enter your Mark: ");
    int mark =System.in.read();   //  returns ASCII code of 1st character
    //Write output to System.out
    System.out.println("Hello, " + name + "!");
    System.out.println("You are " + age + " years old.");
    System.out.println("Your Mark is: " + (char)mark );  //  will print the character
    //Write output to System.err
    System.err.println("This is an error message. Mark is displayed wrongly");
  }
}
```

Output:

```
Enter your name: Arjun
Enter your age: 22
Enter your Mark: 50
Hello, Arjun!
You are 22 years old.
Your Mark is: 5
This is an error message. Mark is displayed wrongly.
```

Explain byte stream and character stream in detail with a example java program.

Stream:

A stream is a flowing sequence of data that a user can either read from or write to

Types of Streams

1. Byte stream

2. character stream

Byte streams:

– provide a convenient means for handling input and output of bytes

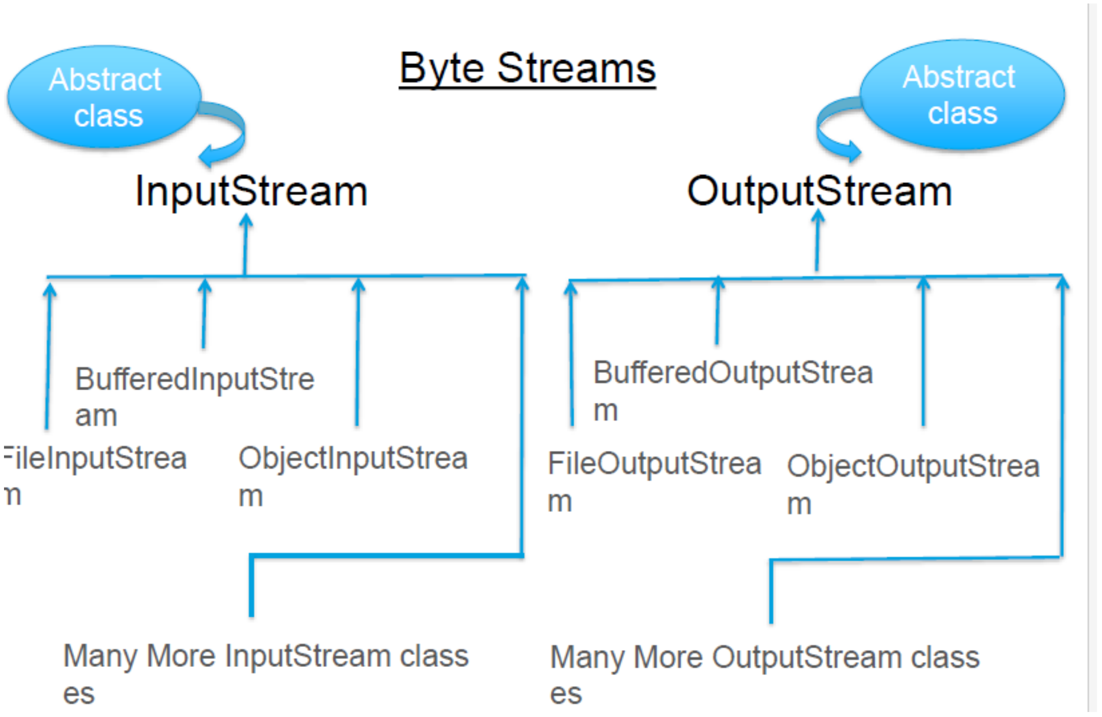
– are used for reading or writing binary data

Character streams:

– provide a convenient means for handling input and output of

characters

– use Unicode, and, therefore, can be internationalized



BufferedINputStream & BufferedOutputStream:

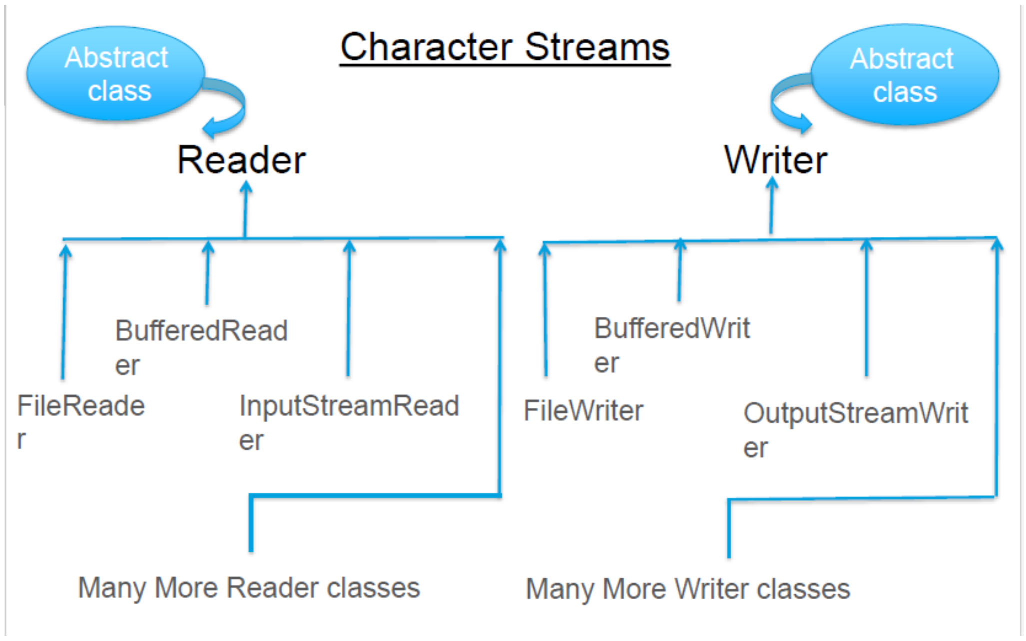
-To read and write data into buffer

FileInputStream & FileOutputStream

-To read and write data into a file

ObjectInputStream & ObjwctOutoutStream:

- To read and write object into secondary device (Serialization)
-



BufferedReader & BufferedWriter:

-To read and write data into buffer

FileIReader & FileWriter

-To read and write data into a file

InputStreamReader & OutputStreamWriter

- Bridge from character stream to byte stream

```
//Java Program to demonstrate byte stream and character stream
import java.io.*;

class StreamDemo {
  public static void main(String args[]) throws IOException {
    //Using byte stream to copy data from xanadu.txt to outbyte.txt
    InputStream inByte = null;
    OutputStream outByte = null;
    try {
      inByte = new FileInputStream("xanadu.txt");
      outByte = new FileOutputStream("outbyte.txt");
      int b;
      while ((b = inByte.read()) != -1) {
        outByte.write(b);
      }
```

```
 } finally {
   if (inByte != null) {
     inByte.close();
   }
   if (outByte != null) {
     outByte.close();
   }
 }
```

//Using character stream to copy data from xanadu.txt to outchar.txt

```
 Reader inChar = null;
 Writer outChar = null;
 try {
   inChar = new FileReader("xanadu.txt");
   outChar = new FileWriter("outchar.txt");
   int c;
   while ((c = inChar.read()) != -1) {
     outChar.write(c);
   }
 } finally {
   if (inChar != null) {
     inChar.close();
   }
   if (outChar != null) {
     outChar.close();
   }
 }
}
```

Explain lambda expression with example

Or

Explain the implementation of functional interface with example

Or

How the lambda expression provides the implementation of an interface? Give suitable example.

Or

With an example demonstrate the following:

i. **Lambda expression with no parameter (6)**
ii. **Lambda expression with single parameter (7)**

Or

Write the syntax and explain lambda expression with / without parameters.

What is Lambda Expression?

- Lambda expressions are like methods and are a short block of code that takes in parameters and returns a value.
- They are anonymous or unnamed method that does not execute on their own.
- it is used to implement a method defined by a functional interface.
- An interface which has only one abstract method is called functional interface.
- Java lambda expression is treated as a function, so compiler does not create .class file.

Syntax

(argument-list) -> {body}

Java lambda expression is consisted of three components.

1) Argument-list: It can be empty or non-empty as well.

2) Arrow-token: It is used to link arguments-list and body of expression.

3) Body: It contains expressions and statements for lambda expression (single line or block of code).

No Parameter Syntax

```
() -> {
//Body of no parameter
lambda
}
```

Single Parameter Syntax

```
(p1) -> {
//Body of single paramete
r lambda
}
```

Multiple Parameter Syntax

```
(p1,p2) -> {
//Body of multiple parame
ter lambda
}
```

Note: The Parameter type can be implicit or it can be specified explicitly

Functional Interface

A functional interface is an interface that specifies only one abstract method.

```
Interface MyNumber

{

        double getValue();

}
```

- the method getValue() is implicitly abstract, and it is the only method defined by MyNumber
- Thus, MyNumber is a functional interface,and its functionis defined by getValue()
- A lambda expression is not executed on its own
- It forms the implementation of the abstract method defined by the functional interface that specifies its target type
- When a lambda expression occurs in a target type context, an instance of a class is automatically created that implements the functional interface, with the lambda expression defining the behavior of the abstract method declared by the functional interface
- When that method is called through the target, the lambda expression is executed
- Inorder for a lambda expression to be used in a target type context, the type of the abstract method and the type of the lambda expression must be compatible

Lamda Extression with no parameter:

```
  import java.util.*;
 interface MyNumber{
doublegetValue();
}
public class LB1 {
public static void main(String[] args) {  MyNumber MyNum;
MyNum = () -> 123.45;
System.out.println("Fixed Value " + MyNum.getValue());
MyNum = () -> Math.random() * 100;
System.out.println("Random Value " + MyNum.getValue());
System.out.println("Another Random Value " + MyNum.getValue());
}
}
```

Output:

```
Fixed Value 123.45
Random Value 78.4020074321781
Another Random Value 15.76123901
```

Lambda Expression with Parameter(Multiple)

```
interface Addable{
        int add(int a,int b);
}

public class LambdaExpressionExample5{
  public static void main(String[] args) {

    // Multiple parameters in lambda expression
    Addable ad1=(a,b)->(a+b);
    System.out.println(ad1.add(10,20));

    // Multiple parameters with data type in lambda expression
    Addable ad2=(int a,int b)->(a+b);
    System.out.println(ad2.add(100,200));
  }
}
```

```
Output:
30
300
```

Note: if the question is asked for lamda with single parameter, change the same program accordingly with single parameter

Lamda Expression with single parameter (Lamda Expression with Block of code)

- The right side of the lambda operator consists of a block of code that can contain more than one Statement
- A block lambda is easy to create. Simply enclose the body within braces as you would any other block of statements
- we must explicitly use a return statement to return a value

Example:

```
interface NumericFunc {
int func(intn);
}
public class LB4{

public static void main(String[] args) {
NumericFunc factorial = (n) ->{
int result =1;
for (int i = 1; i <= n; i++)
        result = i *result;
Return result;
};
System.out.println("The factorial is " +factorial.func(3));
System.out.println("The factorial is " +factorial.func(5));
}
}
```

Output:

```
The factorial is 6
The factorial is 120
```