Mahatma Education Society's

Pillai HOC College of Engineering and Technology, Rasayani

Department of Computer Engineering

Practical List

Subject: Object Oriented Programming Methodology **Semester: III**

H/W Requirements	Desktop Computer, 1GB RAM, 256GB HDD
S/W Requirements	Windows 7 OS, JDK

EXPERIMENT NO: 1

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

Aim: Program on various ways to accept data through keyboard.

Theory:

1. **Using Scanner class**

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them. These predefined classes are organized in the form of packages. A package means collection of classes. Before we can use the class, we need to import the entire package or the class. We import the Scanner class using the following line.

```
import java.util.Scanner;
```

The import statement should be the first line in our program and then as usual we write the code for our class or program. We have to create an object of scanner class by using following syntax.

```
Scanner s = new Scanner ( System.in );
a=s.nextInt();// To accept integer value
b=s.nextFloat();//To accept float value
```

The Scanner class has several methods which are used to take different types of inputs.

Method	**Description**
nextByte()	Accept a byte
nextShort()	Accept a short
nextInt()	Accept a integer
nextLong()	Accept a long
next()	Accept a single word
nextLine()	Accept a multi word string
nextBoolean()	Accept a boolean
nextFloat()	Accept a float
nextDouble()	Accept a double

2. **Using BufferedReader class**

BufferedReader class:

Reads text from the character input stream, buffering characters so as to provide an efficient reading of characters, arrays and lines.

InputStreamReader class:

Reads bytes and decodes them into characters by using specified charset.

By wrapping the **System.in** (standard input stream) in an **InputStreamReader** class which is wrapped in a **BufferedReader**, we can read input from the user in the command line.

Syntax: To accept integer and float values

```
BufferedReader  obj = new BufferedReader(new InputStreamReader(System.in));
a= Integer.parseInt(obj.readLine());
b= Float.valueOf(obj.readLine()).floatValue();
```

3. Command Line Arguments

Sometimes we want to pass information into a program when we run it.

This can be done by passing command line arguments to main().

A command line argument is the information that is followed by name of the program on the command line when it is executed.

The arguments are stored as strings in the string array passes to main().

Syntax: For example

```
 javac Sample .java
 java Sample a b c d
```

(Here a, b, c and d are command line arguments.)

Conclusion:

……………………………………………………………………………………………………
……………………………………………………………………………………………………
…………………………………………………………………………………………

Questions:

1. Explain Scanner class and its methods?

----

2. Explain what is BufferedReader and InputStreamReader ?

----

3. What are command line arguments?

EXPERIMENT NO: 2

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date

& Marks

Aim: Program on branching, looping, labelled break and labelled continue.

Theory:

Branching Statements:

if statement:

Syntax:

```
if( condition)
{
// execute this statement if condition is true
}
```

if-else statement:

Syntax:

```
if( condition)
{
// execute this statement if condition is true
}
else
{
// execute this statement if condition is false
}
```

Nested if else

Syntax:

```
if(condition1)
{
        if(condition2)
        {
        executestatement1;
        }
        else
        {
        execute statement2;
        }

}
        .
        .
        else
        {
        execute statement3;

        }
```

Looping Statements:

1. for loop

Syntax:

```
for(initialization; test condition; inc/dec)
{
//execute statements until condition is true;
}
```

2. while loop

Syntax:

```
while(condition)
{
//execute statements until condition is true;
}
```

3. do-while loop

Syntax:

```
do
{
//execute statements without checking condition;
}while(condition);
```

Labelled break and labelled continue:

You can use labels with break andcontinue. Labels are where you can shift control from break and continue. by default break goes outside of loop but if you more than one loop you can use break statement to break a specific loop by providing label. same is true for continue. if you are working on nested loops labels gives more control to break and continue. In following example of break and continue with labels. we have a nested loop and we have created two labels OUTER and INNER. OUTER label is for outer loop and INNER label is for inner loop. we are iterating through array and print value of odd number from outer loop and then use continue statement, which ignores execution of inner loop. if its even number than inner loop executes and breaks as soon as it prints the number.

Conclusion:
…………………………………………………………………………………………………
…………………………………………………………………………………………………
………………………………………………………………………………………

Questions:

1. Explain branching statements along with its syntax?

----

2. Explain looping statements?

3.Explain the switch statement?

4.Explain labeled break and labeled continue?

EXPERIMENT NO: 3

Name of the Student:-____

Roll No.___________

Date of Practical Performed:-__________ Staff Signature with Date & Marks

Aim: To learn the concept of class, data members, methods and to implement a progam.

2.1 Class

A Class is a collection of objects of similar type. Objects are instances of the Class. Once a class is defined, any number of objects can be created which belongs to that class. A class contains many data members that help justify the object of the class. It also contains methods that help perform actions on the objects.

A general class declaration:

```
class class_Name
{
    //public variable declaration
    Data_member 1;
    Data_member 2;
    //public method definitions
    void method_1()
    {
        //body of method…
    }
    void method_2()
```

```
        {
                //body of method…
        }


}
```

Java is a purely object Oriented language so everything inside a program comes inside a class. The ‘main’ method comes inside another class whose name will also be shared by the name of the program.

```
Class Test
{
        public static void main(String args[])
        {
                Class_Name object=new Class_Name();
                object. method_1();
                object. method_2();
        }


}
```

The three methods are called by using the dot operator. When we call a method the code inside its block is executed.

The dot operator is used to call methods or access them.

Conclusion:
……
……
…………………………………………………………………………………………………

Questions:

1. What is a class and an object?

2.What feature of Java does class implement?

----

3. Write the different uses of the dot operator?

----

4. Explain the use of Scope resolution operator.

----

5. Explain why Java is a purely object oriented language.

6.How can we create an instance of the class ?

EXPERIMENT NO: 4

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

Aim:

Understanding constructor and constructor overloading

Theory:

A constructor is a special method that is used to initialize an object.Every class has a constructor,if we don't explicitly declare a constructor for any java class the compiler builds a default constructor for that class. A constructor does not have any return type.
A constructor has same name as the class in which it resides. Constructor in Java can not be abstract, static, final or synchronized. These modifiers are not allowed for constructor.

There are two types of Constructor

- Default Constructor
- Parameterized constructor

Each time a new object is created at least one constructor will be invoked.

```
Car c = new Car()       //Default constructor invoked
Car c = new Car(name); //Parameterized constructor invoked
```

Constructor Overloading

Like methods, a constructor can also be overloaded. Overloaded constructors are differentiated on the basis of their type of parameters or number of parameters. Constructor overloading is not much different than method overloading. In case of method overloading you have multiple methods with same name but different signature, whereas in Constructor overloading you have multiple constructor with different signature but only difference is that Constructor doesn't have return type in Java. A class can have several constructors. Note that all constructor of a class must have different parameter list i.e the signature of each constructor must differ.The signature of a constructor is a combination of its name and the sequence of its parameter types.Overloaded constructors are very common in java because they provide several ways to create an object of a particular class. It also allows you to create and initialize an object by using different types of data.

```
class clsName
{
      // Variable declaration

Public:

      // Constructors
      clsName1()
      {
         // body of constructor
      }

      clsName2(parameter_list_2)
      {
         // body of constructor
      }
          .

          .
      clsNameN(parameter_list_N)
      {
          // body of constructor
      }

      // Methods
}
```

Conclusion:

....
....
....

Questions:

1. Explain constructor.

2.How is a constructor different from a method

----

3. State the rules for constructors.

4. Do constructors have any return type ?

----

5. Is it mandatory to define constructor in class?

EXPERIMENT NO: 5

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

Aim: To understand the concept of Method Overloading.

Theory:

If two or more method in a class have same name but different signature, it is known as method overloading. Overloading always occur in the same class. It is one of the ways through which java supports polymorphism. Method Overloading is a feature that allows a class to have two or more methods having same name, if their argument lists are different.

Argument lists could differ in –

1. Number of parameters.

2. Data type of parameters.

3. Sequence of Data type of parameters

When an overloaded method is invoked, Java uses the type and/or number of arguments to determine which version of the overloaded method to actually call. The rules of method overloading are as follows:

- Overloaded methods MUST differentiate argument list either number or type of arguments.
- Return type of overloaded methods CAN be same or different. It plays no role in overloading
- Access specifier (public, protected, or private) CAN also be same or different for overloaded methods.

Conclusion:
……………………………………………………………………………………………………
……………………………………………………………………………………………………
…………………………………………………………………………………………

Questions:

1.Explain how method overloading is an example of polymorphism

----

----

2. What happends when an overloaded method is invoked?

----

3. State the rules of method overloading.

4. What is Method in java?

----

5. Can we overload main method in java?

EXPERIMENT NO: 6

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

Aim: To understand the concept of Packages

Theory:

Packages

It is a mechanism to encapsulate a group of classes, interfaces and sub packages. Many implementations of Java use a hierarchical file system to manage source and class files. It is easy to organize class files into packages. All we need to do is put related class files in the same directory, give the directory a name that relates to the purpose of the classes, and add a line to the top of each class file that declares the package name, which is the same as the directory name where they reside.
In java there are already many predefined packages that we use while programming.

Advantages of using a package

Before discussing how to use them Let see why we should use packages.

- Reusability: Reusability of code is one of the most important requirements in the software industry. Reusability saves time, effort and also ensures consistency. A class once developed can be reused by any number of programs wishing to incorporate the class in that particular program.
- Easy to locate the files.
- In real life situation there may arise scenarios where we need to define files of the same name. This may lead to “name-space collisions”. Packages are a way of avoiding “name-space collisions”.

Types of package:

1) User defined package: The package we create is called user-defined package.

2) Built-in package: The already defined package like java.io.*, java.lang.* etc are known as built-in packages.

Defining a Package:

This statement should be used in the beginning of the program to include that program in that particular package.

package <package name>;

Compiling packages in java:

The java compiler can place the byte codes in a directory that corresponds to the package declaration of the compilation unit. The java byte code for all the classes(and interfaces) specified in the source files myClass1.java and myClass2.java will be placed in the directory named myPackage1/myPackage2 , as these sources have the following package declaration package myPackage1.myPackage2;

The absolute path of the myPackage1/myPackage2 directory is specified by using the –d (destination directory) option when compiling with the javac compiler.

Assume that the current directory is /packages/project and all the source files are to be found here,the command,

```
javac –d .file1.java file2.java
```

How do we run the program?

Since the current directory is /packages/project and we want to run file1.java,the fully qualified name of the file1 class must be specified in the java command,

```
java myPackage1.myPackage2.file1
```

Conclusion:

……………………………………………………………………………………………………
……………………………………………………………………………………………………
………………………………………………………………………………………………

Questions:

1. What are packages? What are its advantages?

----

2. Which package is always imported by default?

----

3. What is the package name for Exception , Error and Throwable classes?

4. What do you understand by package access specifier?

EXPERIMENT NO: 7

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

Aim: To understand the concept of One dimensional and two dimensional array

Theory: An array is a group of same kind of variables and can be accessible by a common name. Any element in the array can be accessed by its index. Arrays can be one dimensional or multi-dimensional

One dimensional array:

One dimensional array is a list of same typed variables. To create an array, first you must declare an array variable of required type, the syntax is given below:

```
type variable-name[];
```

Example: int account_numbers[];

To allocate memory or create array object the syntax is given below:

variable-name = new type[size of the array];

eg: account_numbers = new int[10];

You must specify the size of the array during creating object.

Multi dimensional array:

Multi dimensional arrays are arrays of arrays. To declare a multi dimensional array variable, specify each additional index using another set of square brackets. An example is given below for two dimensional array:

int sample[][] = new int[3][3];

We can specify memory for first dimension first, and we can assign memory for second dimension later as given : int sample[][] = new int[3][];

Conclusion:

……………………………………………………………………………………………………
……………………………………………………………………………………………………
…………………………………………………………………………………………

Questions:

1.What is an array? What are the types of arrays?

----

2. How to declare a two dimensional array?

----

3. How the individual elements of an array can be accessed?

4. Explain memory allocation in arrays

----

5. Can you change size of array once created?

EXPERIMENT NO: 8

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

Aim: To understand the String and String Buffer class

Theory:

The java.lang.String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.Strings are constant, their values cannot be changed after they are created.

Class constructors

String() This initializes a newly created String object so that it represents an empty character sequence.
String(char[] value) This allocates a new String so that it represents the sequence of characters currently contained in the character array argument.
String(char[] value, int offset, int count) This allocates a new String that contains characters from a subarray of the character array argument.
String(String original) This initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

String(StringBuffer buffer) This allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

char charAt(int index) This method returns the char value at the specified index.
int compareTo(String anotherString) This method compares two strings lexicographically.
int compareToIgnoreCase(String str) This method compares two strings lexicographically, ignoring case differences.
String concat(String str) This method concatenates the specified string to the end of this string.
boolean contains(CharSequence s) This method ceturns true if and only if this string contains the specified sequence of char values.
static String copyValueOf(char[] data) This method returns a String that represents the character sequence in the array specified.
boolean endsWith(String suffix) This method tests if this string ends with the specified suffix.

boolean equals(Object anObject)	This method compares this string to the specified object.
boolean equalsIgnoreCase(String anotherString)	This method compares this String to another String, ignoring case considerations.
int indexOf(int ch)	This method returns the index within this string of the first occurrence of the specified character.
int indexOf(int ch, int fromIndex)	This method returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int indexOf(String str)	This method returns the index within this string of the first occurrence of the specified substring.
int indexOf(String str, int fromIndex)	This method returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
boolean isEmpty()	This method returns true if, and only if, length() is 0.
int lastIndexOf(int ch)	This method returns the index within this string of the last occurrence of the specified character.
int lastIndexOf(String str)	This method returns the index within this string of the rightmost occurrence of the

specified substring.
int lastIndexOf(String str, int fromIndex) This method returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int length() This method returns the length of this string.
String replace(char oldChar, char newChar) This method returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
String replace(CharSequence target, CharSequence replacement) This method replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.
String replaceAll(String regex, String replacement) This method replaces each substring of this string that matches the given regular expression with the given replacement.
String replaceFirst(String regex, String replacement) This method replaces the first substring of this string that matches the given regular expression with the given replacement.
String substring(int beginIndex) This method returns a new string that is a substring of this string.
String substring(int beginIndex, int endIndex) This method returns a new string that is a substring of this string.

char[] toCharArray() This method converts this string to a new character array.
String toLowerCase() This method converts all of the characters in this String to lower case using the rules of the default locale.
String toLowerCase(Locale locale) This method converts all of the characters in this String to lower case using the rules of the given Locale.
String toString() This method returns the string itself.
String toUpperCase() This method converts all of the characters in this String to upper case using the rules of the default locale.
String toUpperCase(Locale locale) This method converts all of the characters in this String to upper case using the rules of the given Locale.
String trim() This method returns a copy of the string, with leading and trailing whitespace omitted.

STRINGBUFFER Class

The java.lang.StringBuffer class is a thread-safe, mutable sequence of characters. Following are the important points about StringBuffer −

- A string buffer is like a String, but can be modified.

- It contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.
- They are safe for use by multiple threads.
- Every string buffer has a capacity.

Class constructors

S.N.	Constructor & Description
1	StringBuffer() This constructs a string buffer with no characters in it and an initial capacity of 16 characters.
2	StringBuffer(CharSequence seq) This constructs a string buffer that contains the same characters as the specified CharSequence.
3	StringBuffer(int capacity) This constructs a string buffer with no characters in it and the specified initial capacity.
4	StringBuffer(String str) This constructs a string buffer initialized to the contents of the specified string.

Class methods

Method & Description
StringBuffer append(boolean b) This method appends the string representation of the boolean argument to the sequence

StringBuffer append(char c) This method appends the string representation of the char argument to this sequence.
StringBuffer append(char[] str) This method appends the string representation of the char array argument to this sequence.
StringBuffer append(CharSequence s) This method appends the specified CharSequence to this sequence.
StringBuffer append(CharSequence s, int start, int end) This method appends a subsequence of the specified CharSequence to this sequence.
StringBuffer append(double d) This method appends the string representation of the double argument to this sequence.
StringBuffer append(float f) This method appends the string representation of the float argument to this sequence.
StringBuffer append(String str) This method appends the specified string to this character sequence.
int capacity() This method returns the current capacity.
char charAt(int index)

This method returns the char value in this sequence at the specified index.
StringBuffer delete(int start, int end) This method removes the characters in a substring of this sequence.
StringBuffer deleteCharAt(int index) This method removes the char at the specified position in this sequence
int indexOf(String str) This method returns the index within this string of the first occurrence of the specified substring.
int indexOf(String str, int fromIndex) This method returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
StringBuffer insert(int offset, boolean b) This method inserts the string representation of the boolean argument into this sequence.
StringBuffer insert(int offset, char c) This method inserts the string representation of the char argument into this sequence.
StringBuffer insert(int offset, char[] str) This method inserts the string representation of the char array argument into this sequence.
StringBuffer insert(int index, char[] str, int offset, int len) This method inserts the string representation of a subarray of the str array

argument into this sequence.
int lastIndexOf(String str) This method returns the index within this string of the rightmost occurrence of the specified substring.
int lastIndexOf(String str, int fromIndex) This method returns the index within this string of the last occurrence of the specified substring.
int length() This method returns the length (character count).
StringBuffer replace(int start, int end, String str) This method replaces the characters in a substring of this sequence with characters in the specified String.
StringBuffer reverse() This method causes this character sequence to be replaced by the reverse of the sequence.
void setCharAt(int index, char ch) The character at the specified index is set to ch.
void setLength(int newLength) This method sets the length of the character sequence.
CharSequence subSequence(int start, int end) This method returns a new character sequence that is a subsequence of this sequence.

String substring(int start) This method returns a new String that contains a subsequence of characters currently contained in this character sequence
String substring(int start, int end) This method returns a new String that contains a subsequence of characters currently contained in this sequence.
String toString() This method returns a string representing the data in this sequence.
void trimToSize() This method attempts to reduce storage used for the character sequence.

Conclusion:
……
……
……………………………………………………………………………………………………

Questions:

1. Is *String* a keyword in java?

----

2. Is *String* a primitive type or derived type?

----

3. What is the main difference between Java strings and C, C++ strings?

4. What is the difference between String and StringBuffer class?

----

5. What is difference between equal() and == ?

EXPERIMENT NO: 9

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

Aim:

To understand the concept of Vector and how it is different from Arrays.

Theory:

We know that Arrays are very useful when there is a need to use number of variables. But there is a problem with array they use only single data type or the elements of array are always same type for avoiding this problem vectors are used.These are also collection of elements those are object data type for using vectors we have to import java.util package.These are also called as dynamic array of object data type.

But Vectors doesn’t support primitives data types like int, float, char etc. For creating a vector, vector class will be used which resides in java’s utility package.

The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

Each vector tries to optimize storage management by maintaining a capacity and a capacityIncrement. The capacity is always at least as large as the vector size; it is usually larger because as components are added to the vector, the vector's storage increases in chunks the size of capacityIncrement. An application can increase the capacity of a vector before inserting a large number of components; this reduces the amount of incremental reallocation.

Three ways to create vector class object:

Method 1:

Vector vec = new Vector();

It creates an empty Vector with the default initial capacity of 10. It means the Vector will be re-sized when the 11th elements needs to be inserted into the Vector. Note: By default vector doubles its size. i.e. In this case the Vector size would remain 10 till 10 insertions and once we try to insert the 11th element It would become 20 (double of default capacity 10).

Method 2:

Syntax: Vector object= new Vector(int initialCapacity)

Vector vec = new Vector(3);

It will create a Vector of initial capacity of 3.

Method 3:

Syntax:

Vector object= new vector(int initialcapacity, capacityIncrement)

Vector vec= new Vector(4, 6)

Here we have provided two arguments. The initial capacity is 4 and capacityIncrement is 6. It means upon insertion of 5th element the size would be 10 (4+6) and on 11th insertion it would be 16(10+6).

Important methods of Vector Class:

- void addElement(Object element): It inserts the element at the end of the Vector.
- int capacity(): This method returns the current capacity of the vector.
- int size(): It returns the current size of the vector.
- void setSize(int size): It changes the existing size with the specified size.
- boolean contains(Object element): This method checks whether the specified element is present in the Vector. If the element is been found it returns true else false.
- boolean containsAll(Collection c): It returns true if all the elements of collection c are present in the Vector.
- Object elementAt(int index): It returns the element present at the specified location in Vector.
- Object firstElement(): It is used for getting the first element of the vector.
- Object lastElement(): Returns the last element of the array.
- Object get(int index): Returns the element at the specified index.
- boolean isEmpty(): This method returns true if Vector doesn’t have any element.
- boolean removeElement(Object element): Removes the specifed element from vector.
- boolean removeAll(Collection c): It Removes all those elements from vector which are present in the Collection c.
- void setElementAt(Object element, int index): It updates the element of specifed index with the given element.

Conclusion:

…………………………………………………………………………………………………
…………………………………………………………………………………………………
…………………………………………………………………………………………

Questions:

1. What is a vector?

2.State the different methods for creating a vector.

----

3. How is a vector different from an Array.

EXPERIMENT NO: 10

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

Aim: **To understand inheritance**

Theory:

Inheritance is one of the feature of Object-Oriented Programming (OOPs). Inheritance allows a class to use the properties and methods of another class. In other words, the derived class inherits the states and behaviors from the base class. The derived class is also called subclass and the base class is also known as super-class. The derived class can add its own additional variables and methods. These additional variable and methods differentiates the derived class from the base class. Inheritance is a compile-time mechanism. A super-class can have any number of subclasses. But a subclass can have only one superclass. This is because Java does not support multiple inheritance. The superclass and subclass have **"is-a"** relationship between them. Let's have a look at the example below.

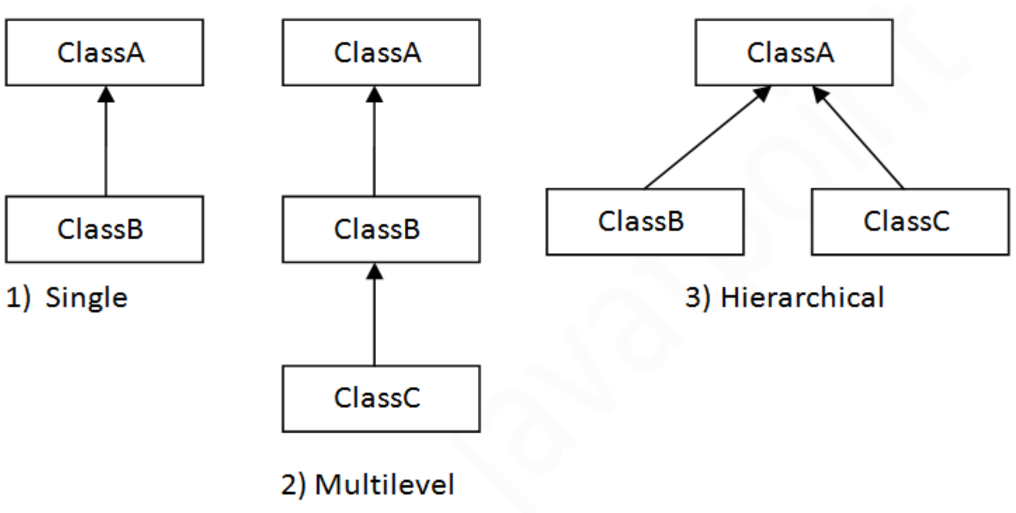
Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality. In the terminology of Java, a class which is inherited is called parent or super class and the new class is called child or subclass.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.Multiple inheritance is not supported in java through class. For implementing Multiple inheritance, Interfaces are used.



Inheritance and constructors in Java

In Java, constructor of base class with no argument gets automatically called in derived class constructor. For example, output of following program is:

```
Base Class Constructor Called
Derived Class Constructor Called
```

Syntax:

```
 class Base() {
    System.out.println("Base Class Constructor Called ");
  }
}

class Derived extends Base {
  Derived() {
    System.out.println("Derived Class Constructor Called ");
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    Derived d = new Derived();
  }
}
```

But, if we want to call parameterized contructor of base class, then we can call it using super(). The point to note is base class comstructor call must be the first line in derived class constructor. For example, in the following program, super(_x) is first line derived class constructor.

```
// filename: Main.java

class Base {

  int x;

  Base(int _x) {

    x = _x; }

}

 class Derived extends Base {

  int y;

  Derived(int _x, int _y) {

    super(_x);

    y = _y;

  }

  void Display() {

    System.out.println("x = "+x+", y = "+y);

  }

}
```

```
public class Main {

  public static void main(String[] args) {

    Derived d = new Derived(10, 20);

    d.Display();

  }

}
```

Output:
```
x = 10, y = 20
```

Conclusion:

……………………………………………………………………………………………………
……………………………………………………………………………………………………
……………………………………………………………………………………………

Questions:

1.Explain inheritance and its advantages.

2.Write the syntx of inheriting a class.

----

3. Explain why Java doesnot support multiple inheritance.

4. Can a class extend more than one classes or does java support multiple inheritance

----

EXPERIMENT NO: 11

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

Aim: Program on abstract class

Theory:

There are situations in which you will want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method. That is, sometimes you will want to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. Such a class determines the nature of the methods that the subclasses must implement. One way this situation can occur is when a superclass is unable to create a meaningful implementation for a method. Any class that contains one or more abstract methods must also be declared abstract. To declare a class abstract, you simply use the abstract keyword in front of the class keyword at the beginning of the class declaration. There can be no objects of an abstract class. That is, an abstract class cannot be directly instantiated with the new operator. Such objects would be useless, because an abstract class is not fully defined. Also, you cannot declare abstract constructors, or abstract static methods. Any subclass of an abstract class must either implement all of the abstract methods in the superclass, or be itself declared abstract

Syntax:

```
abstract class class_name
{
//abstract method decalaration
}
```

Conclusion:

……………………………………………………………………………………………………
……………………………………………………………………………………………………
…………………………………………………………………………………………

Questions:

1.What is an abstract class?

----

2. Explain use of final keyword?

3.Explain access modifiers in java?

EXPERIMENT NO: 12

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

Aim: To understand the concept of Interface

Theory:

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements. An interface can extend multiple interfaces.

An interface is similar to a class in the following ways:

- An interface can contain any number of methods.
- An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
- The bytecode of an interface appears in a .class file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including:

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class..

When a class implements an interface, the class has to define all the methods of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Syntax

class derived_class_name extends base_class_name implements Interface_name

Eg: Class **Lion** extends **Animal** implements **Mammal**

Conclusion:

……………………………………………………………………………………………………
……………………………………………………………………………………………………
……………………………………………………………………………………………

Questions:

1. What is an interface in Java

----

2. What will happen if we define a concrete method in an interface

----

----

3. Can we create non static variables in an interface?

4. When we need to use extends and implements?

----

5. Can we create object for an interface?

EXPERIMENT NO: 13

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

AIM:
To understand the concept of Exceptions and learn Exception handling

THEORY:

An Exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

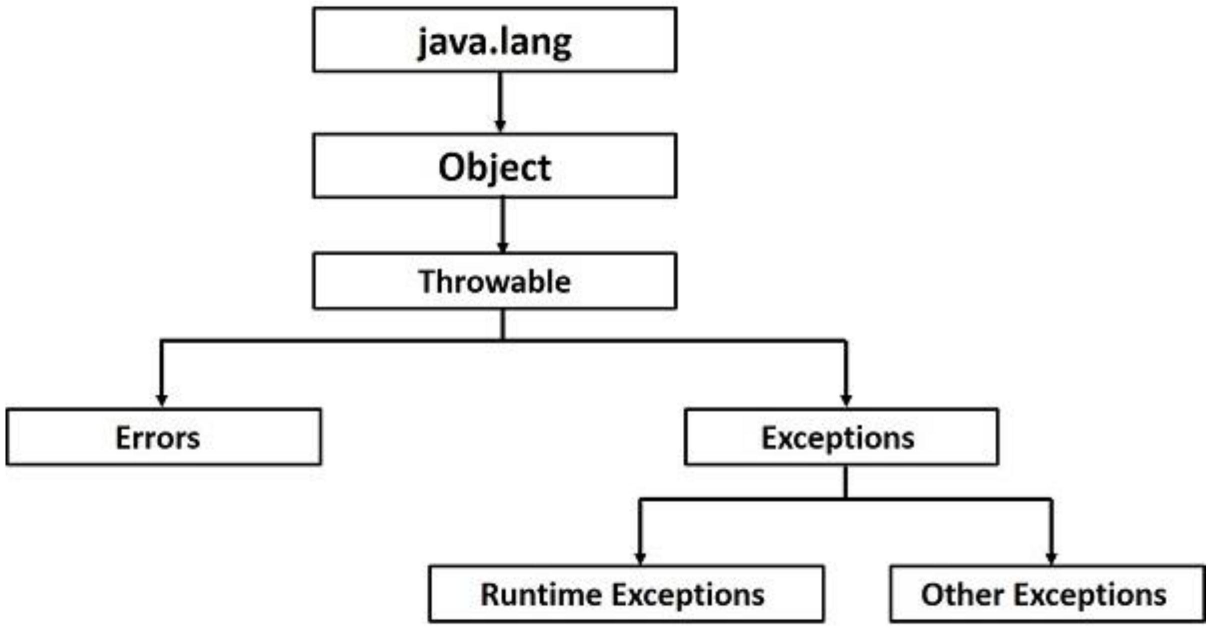
Based on these, we have three categories of Exceptions

- **Checked exceptions** − A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.

- **Unchecked exceptions** − An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

Exception Hierarchy

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class.The Exception class has two main subclasses: IOException class and RuntimeException Class.



Catching Exceptions

A method catches an exception using a combination of the try and catchkeywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following –

```
try {
   // Protected code
}catch(ExceptionName e1) {
   // Catch block
}
```

If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature.You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the throw keyword.

The difference between throw and throws?

throw keyword	throws keyword
checked exceptions can not be propagated with throw only.	checked exception can be propagated with throws.
throw is followed by an instance.	throws is followed by class.
throw is used within the method.	throws is used with the method signature.

The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

```
try {
   // Protected code
}catch(ExceptionType1 e1) {
   // Catch block
}catch(ExceptionType2 e2) {
   // Catch block
}catch(ExceptionType3 e3) {
   // Catch block
}finally {
   // The finally block always executes.
}
```

Note the following −

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block exists
- The try block cannot be present without either catch clause or finally clause.
- Any code cannot be present in between the try, catch, finally blocks.

User-defined Exceptions

You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes −

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

We can define our own Exception class as below −

class MyException extends Exception {}

You just need to extend the predefined Exception class to create your own Exception. These are considered to be checked exceptions. An exception class is like any other class, containing useful fields and methods.

Conclusion:

……………………………………………………………………………………………………
……………………………………………………………………………………………………
………………………………………………………………………………………………

1. What is exception? How different is it from an Error?

----

2. What are run time exceptions in java.

----

3. What is the difference between final, finally and finalize in java?

----

4. State the differences between throw and throws

EXPERIMENT NO: 14

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

AIM: To understand the concept of threading by extending Thread Class.

THEORY:

Threading is a facility to allow multiple tasks to run concurrently within a single process. Threads are independent, concurrent execution through a program, and each thread has its own stack. In Java threads can be implemented in two ways.

- ***'Extending Thread Class'*** and
- ***'Implementing Runnable Interface***

Extending Thread Class is required to ***override run()*** method. The run method contains the actual logic to be executed by thread. Creation of thread object never starts execution, we need to call ***'start()'*** method to run a thread.Other methods supported by Threads are given below

- ***join():*** It makes to wait for this thread to die. You can wait for a thread to finish by calling its join() method.
- ***sleep():*** It makes current executing thread to sleep for a specified interval of time. Time is in milli seconds.
- ***yield():*** It makes current executing thread object to pause temporarily and gives control to other thread to execute.
- ***notify():*** This method is inherited from Object class. This method wakes up a single thread that is waiting on this object's monitor to acquire lock.
- ***notifyAll():*** This method is inherited from Object class. This method wakes up all threads that are waiting on this object's monitor to acquire lock.
- ***wait():*** This method is inherited from Object class. This method makes current thread to

Conclusion:

………………………………………………………………………………………………
………………………………………………………………………………………………
………………………………………………………………………………………………
………………

1. What is thread ?

----

2. What is the difference between preemptive scheduling and time slicing?

----

3. What is difference between wait() and sleep() method?

EXPERIMENT NO: 15

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

AIM:
To understand the concept of Exceptions and learn Exception handling

THEORY:

Because multithreading introduces an asynchronous behavior to your programs, there must be a way for you to enforce synchronicity when you need it. For example, if you want two threads to communicate and share a complicated data structure, such as a linked list, you need some way to ensure that they don't conflict with each other. That is, you must prevent one thread from writing data while another thread is in the middle of reading it.

For this purpose, Java implements an elegant twist on an age-old model of interprocess synchronization: the monitor. You can think of a monitor as a very small box that can hold only one thread. Once a thread enters a monitor, all other threads must wait until that thread exits the monitor. In this way, a monitor can be used to protect a shared asset from being manipulated by more than one thread at a time. In java, There is no class "Monitor"; instead, each object has its own implicit monitor that is automatically entered when one of the object's synchronized methods is called.

Once a thread is inside a synchronized method, no other thread can call any other synchronized method on the same object. When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.

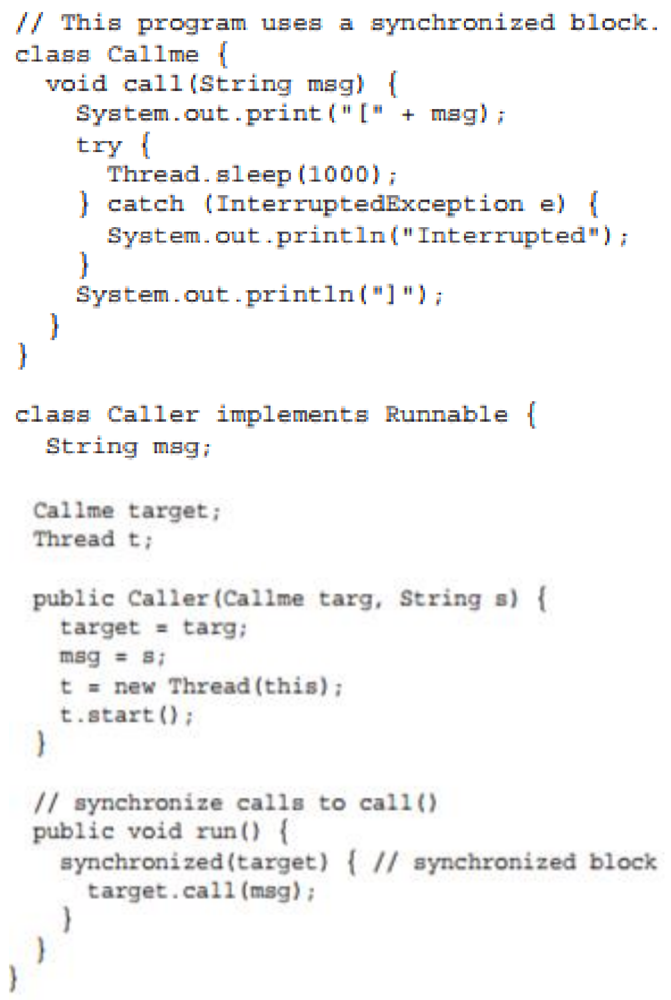
Key to synchronization is the concept of the monitor (also called a semaphore). A monitor is an object that is used as a mutually exclusive lock, or mutex. Only one thread can own a monitor at a given time. When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor. These other threads are said to be waiting for the monitor. A thread that owns a monitor can reenter the same monitor if it so desires. To enter an object’s monitor, just call a method that has been modified with the synchronized keyword.

While a thread is inside a synchronized method, all other threads that try to call it (or any other synchronized method) on the same instance have to wait wait until another thread invokes the notify() or the notifyAll() for this object

Syntax

```
synchronized(objectidentifier) {
   // Access shared variables and other shared resources
}
```

Here, the objectidentifier is a reference to an object whose lock associates with the monitor that the synchronized statement represents.



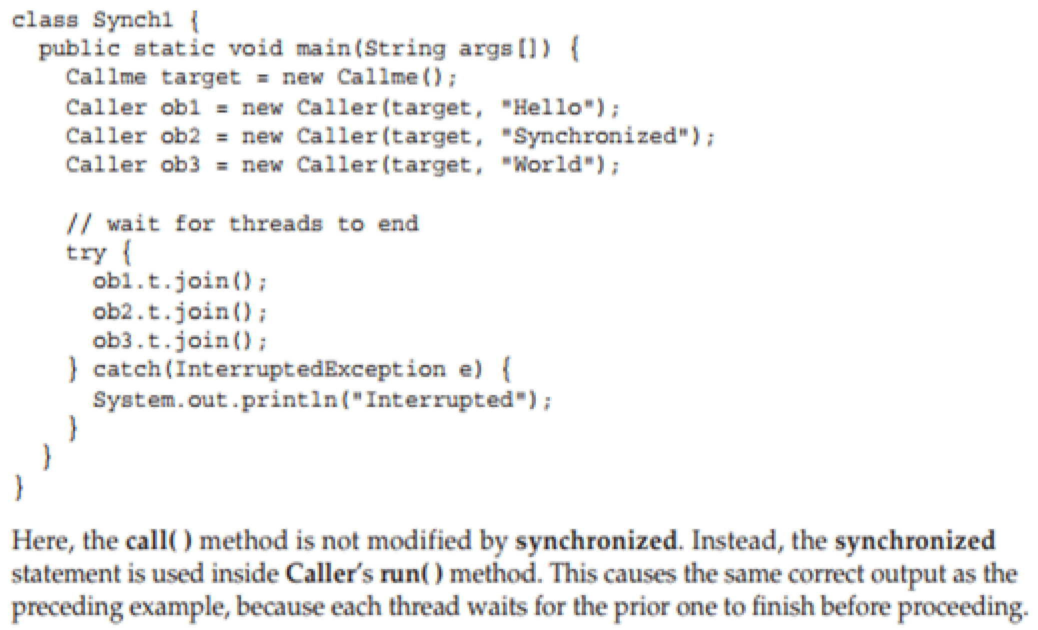
```
// This program uses a synchronized block.
class Callme {
  void call(String msg) {
    System.out.print("[" + msg);
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      System.out.println("Interrupted");
    }
    System.out.println("]");
  }
}

class Caller implements Runnable {
  String msg;

  Callme target;
  Thread t;

  public Caller(Callme targ, String s) {
    target = targ;
    msg = s;
    t = new Thread(this);
    t.start();
  }

  // synchronize calls to call()
  public void run() {
    synchronized(target) { // synchronized block
      target.call(msg);
    }
  }
}
```



```
class Synch1 {
  public static void main(String args[]) {
    Callme target = new Callme();
    Caller ob1 = new Caller(target, "Hello");
    Caller ob2 = new Caller(target, "Synchronized");
    Caller ob3 = new Caller(target, "World");

    // wait for threads to end
    try {
      ob1.t.join();
      ob2.t.join();
      ob3.t.join();
    } catch(InterruptedException e) {
      System.out.println("Interrupted");
    }
  }
}
```

Here, the **call( )** method is not modified by **synchronized**. Instead, the **synchronized** statement is used inside **Caller**'s **run( )** method. This causes the same correct output as the preceding example, because each thread waits for the prior one to finish before proceeding.

Conclusion:

………………………………………………………………………………………………………
………………………………………………………………………………………………………
…………………………………………………………………………………………

1. What is Synchronization of threads?

2.What are synchronized method and Synchronised statement?

----

3. Explain the concept of monitor?

4.Can a constructor be synchronized in Java?

EXPERIMENT NO: 16

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

AIM:
To understand the concept of Applet

THEORY:

- Applets are small Java applications that can be accessed on an Internet server, transported over Internet, and can be automatically installed and run as apart of a web document.
- After a user receives an applet, the applet can produce a graphical user interface. It has limited access to resources so that it can run complex computations without introducing the risk of viruses or breaching data integrity.
- Any applet in Java is a class that extends the java.applet.Applet class.
- An Applet class does not have any main() method. It is viewed using JVM. The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application.

JVM creates an instance of the applet class and invokes init() method to initialize an Applet. Every Applet application must import two packages - **java.awt** and **java.applet**.

- java.awt.* imports the Abstract Window Toolkit (AWT) classes. Applets interact with the user (either directly or indirectly) through the AWT. The AWT contains support for a window-based, graphical user interface.
- java.applet.* imports the applet package, which contains the class Applet. Every applet that you create must be a subclass of Applet class.

The class in the program must be declared as public, because it will be accessed by code that is outside the program.Every Applet application must declare a paint() method. This method is defined by AWT class and must be overridden by the applet. The paint() method is called each time when an applet needs to redisplay its output.

Another important thing to notice about applet application is that, execution of an applet does not begin at main() method. In fact an applet application does not have any main() method.

An Applet Skeleton

Most applets override these four methods. These four methods forms Applet lifecycle.

- init() : init() is the first method to be called. This is where variable are initialized. This method is called only once during the runtime of applet.
- start() : start() method is called after init(). This method is called to restart an applet after it has been stopped.
- stop() : stop() method is called to suspend thread that does not need to run when applet is not visible.
- destroy() : destroy() method is called when your applet needs to be removed completely from memory

Running Applet using Applet Viewer

To execute an Applet with an applet viewer, write short HTML file as discussed above. If you name it as run.htm, then the following command will run your applet program.

```
F:/>appletviewer run.htm
```

Subroutines

In Java, the built-in drawing subroutines are found in objects of the class Graphics, one of the classes of the *java.awt*package. In an applet's paint() routine, you can use the Graphics object g for drawing. This object is provided as a parameter to the paint() routine when the system calls the paint() method. Amongst others, three of theGraphics object's subroutines are:

- **g.setColor(c)** is called to set the color that is used in drawing. The parameter c is an object belonging to the Color class: "g.setColor(Color.RED)".
- **g.drawRect(x, y, w, h)** draws the outline of a rectangle. The parameters x, y, w, h must be integer-valued expressions. This subroutine draws an outline of a rectangle whose top left corner is x pixels from the left edge of the applet. The width of the rectangle is w pixels, and the height is h pixels.

- **g.fillRect(x, y, w, h)** is similar to drawRect except that it fills the inside of the rectangle instead of just drawing an outline.
- **g.drawOval(x, y, w, h)** :draws oval shape

Conclusion:

…………………………………………………………………………………………………
…………………………………………………………………………………………………
…………………………………………………………………………………………

1. What is an applet? What are its advantages?

----

2. Explain the life cycle of an applet.

----

3. Which classes can an applet extend?

----

4. Explain the life cycle of an applet.

EXPERIMENT NO: 17

Name of the Student:-____

Roll No.____________

Date of Practical Performed:-___________ Staff Signature with Date & Marks

AIM:
To understand the concept of Applet

THEORY:

Java allows users to pass user-defined parameters to an applet with the help of <PARAM>tags. The <PARAM>tag has a NAME attribute which defines the name of the parameter and a VALUE attribute which specifies the value of the parameter. In the applet source code, the applet can refer to the parameter by its NAME to find its value. The syntax of the <PARAM>tag is:

```
<APPLET>
<PARAMNAME=parameter1_name VALUE=parameter1_value>
<PARAMNAME=parameter2_name VALUE=parameter2_value>
</APPLET>
```

For example, consider the following statements to set the text attribute of applet to This is an example of Parameter! ! !

```
<APPLET>
 <PARAMNAME=text VALUE=This is an example of Parameter!!!>
 </APPLET>
```

The <PARAM>tags must be included between the <APPLET> and</ APPLET> tags. The init () method in the applet retrieves user-defined values of the parameters defined in the <PARAM>tags by using the get Parameter () method. This method accepts one string argument that holds the name of the parameter and returns a string which contains the value of that parameter. Since it returns String object, any data type other than String must be converted into its corresponding data type before it can be used.

Compile the program :

javac appletParameter.java

Output after running the program :

To run the program using appletviewer, go to command prompt and type appletviewer appletParameter.html Appletviewer will run the applet for you and and it should show output like Welcome in Passing parameter in java applet example. Alternatively you can also run this example from your favorite java enabled browser.

Conclusion:

…………………………………………………………………………………………………
…………………………………………………………………………………………………
…………………………………………………………………………………………

1. Explain about the parameter tag in Applet

----

2. Explain the procedureto pass parameter from an html page to an applet

----

----

3. How will you execute the program? Explain with steps

EXPERIMENT NO: 18

Name of the Student:-____

Roll No.____________

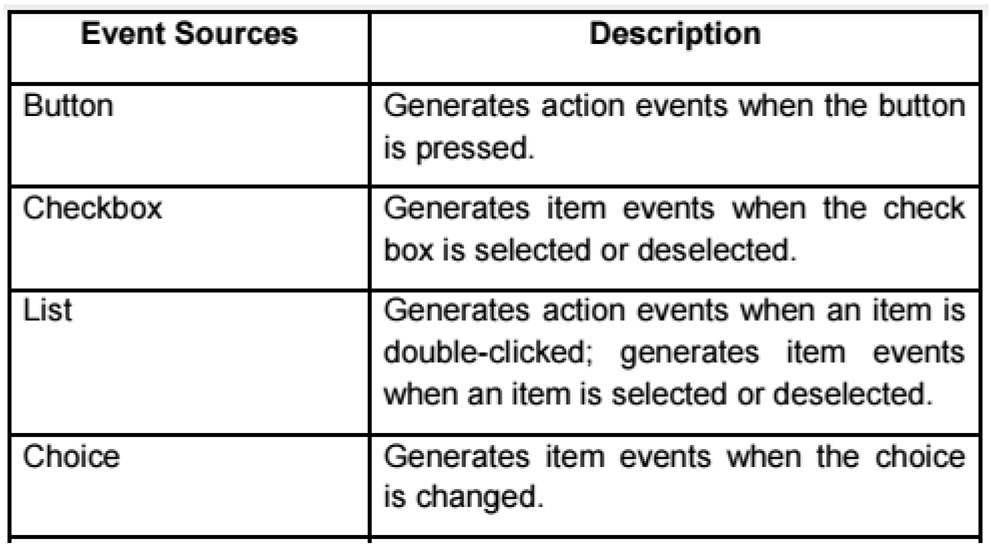
Date of Practical Performed:-___________ Staff Signature with Date & Marks

AIM:

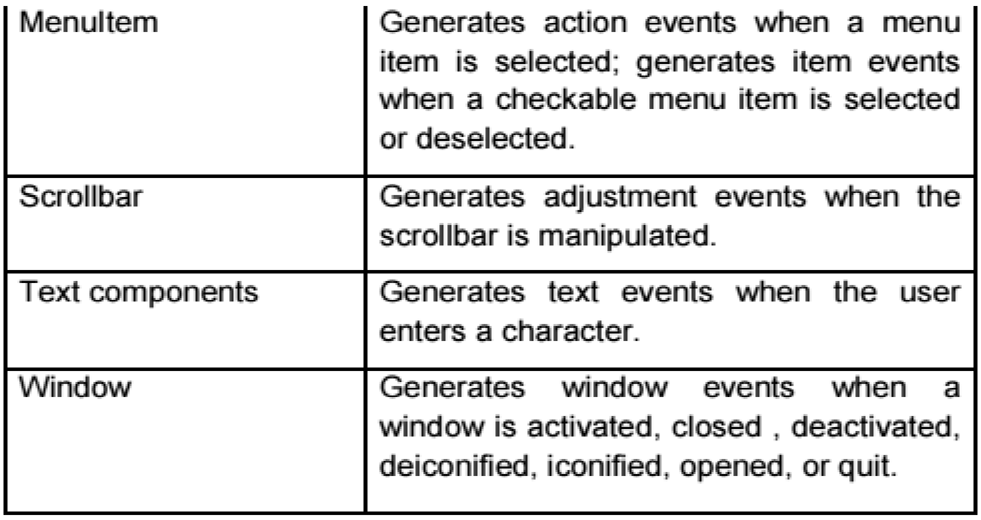
To create GUI application with event handling using AWT controls

THEORY:

Writing an applet that responds to user input, introduces us to event handling. We can make our applet respond to user input by overriding event handler methods in our applet. An Event is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a GUI. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse. An event source is the object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source may register listeners in order for the listeners to receive notifications about a specific type of event.

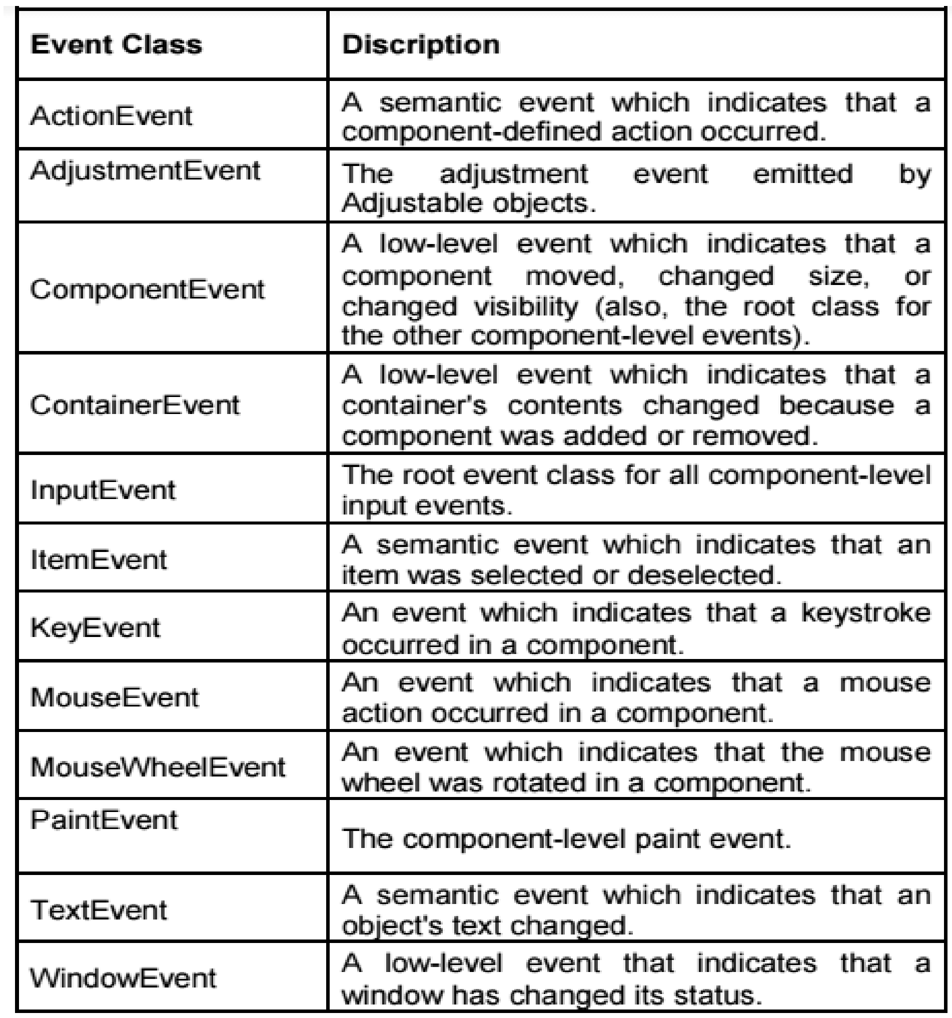


Event Sources	Description
Button	Generates action events when the button is pressed.
Checkbox	Generates item events when the check box is selected or deselected.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Choice	Generates item events when the choice is changed.



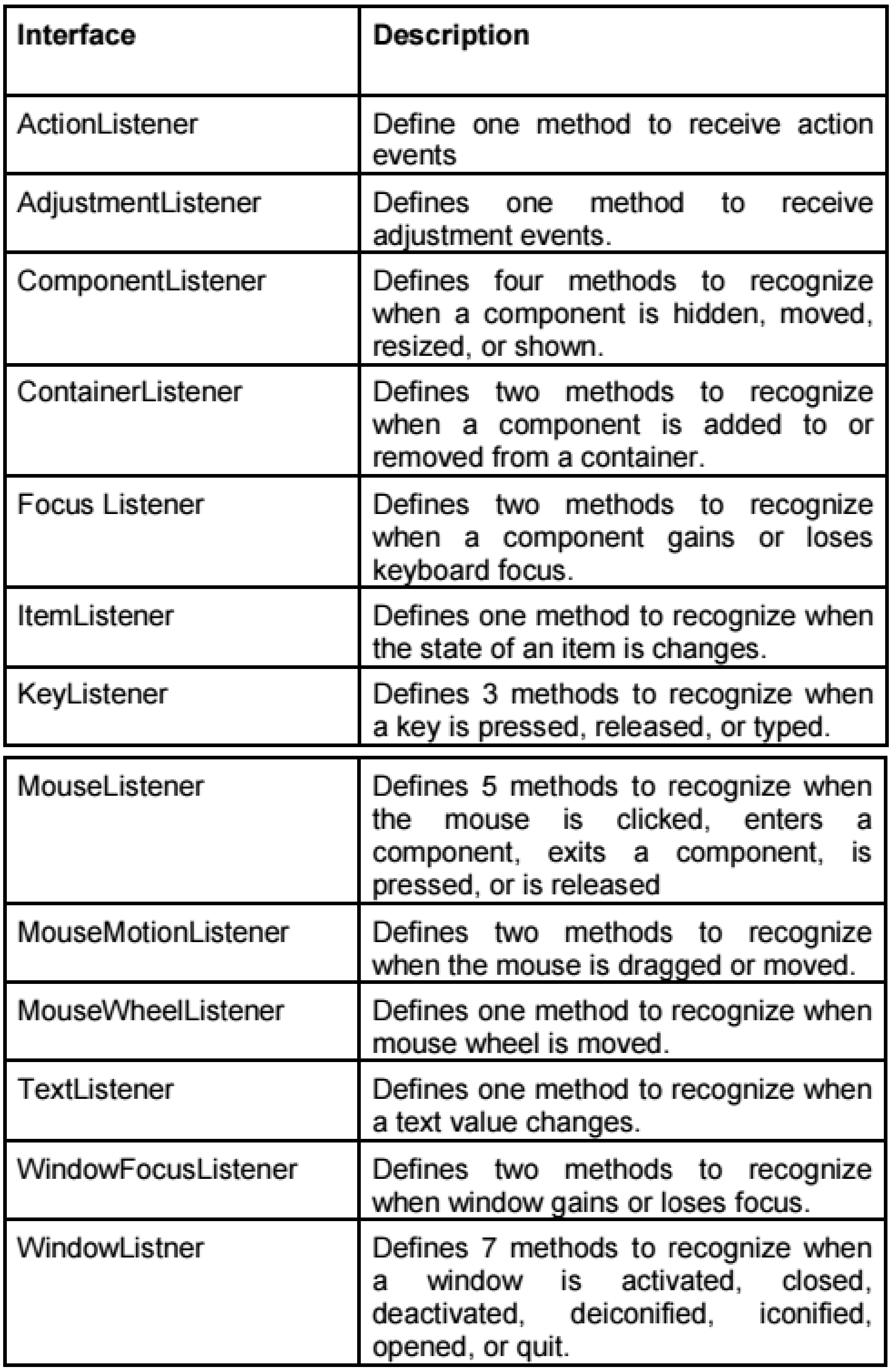
MenuItem	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scrollbar	Generates adjustment events when the scrollbar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed , deactivated, deiconified, iconified, opened, or quit.

Each type of event has its own registration method. The ‘EventObject’ class is at the top of the event class hierarchy. It belongs to the java.util package. While most of the other event classes are present in java.awt.event package. These are objects that define methods to handle certain type of events.



Event Class	**Discription**
ActionEvent	A semantic event which indicates that a component-defined action occurred.
AdjustmentEvent	The adjustment event emitted by Adjustable objects.
ComponentEvent	A low-level event which indicates that a component moved, changed size, or changed visibility (also, the root class for the other component-level events).
ContainerEvent	A low-level event which indicates that a container's contents changed because a component was added or removed.
InputEvent	The root event class for all component-level input events.
ItemEvent	A semantic event which indicates that an item was selected or deselected.
KeyEvent	An event which indicates that a keystroke occurred in a component.
MouseEvent	An event which indicates that a mouse action occurred in a component.
MouseWheelEvent	An event which indicates that the mouse wheel was rotated in a component.
PaintEvent	The component-level paint event.
TextEvent	A semantic event which indicates that an object's text changed.
WindowEvent	A low-level event that indicates that a window has changed its status.

An event source (for example a PushButton) can generate one or more type of events, and maintain a list of event listeners for each type of event. An event source can register listeners by calling addXListener type of methods.



Interface	Description
ActionListener	Define one method to receive action events
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
Focus Listener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item is changes.
KeyListener	Defines 3 methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines 5 methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when window gains or loses focus.
WindowListner	Defines 7 methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Conclusion:

……………………………………………………………………………………………………
……………………………………………………………………………………………………
……………………………………………………………………………………………

1.Which class is the super class of all events.

----

2. What is an event handler?

----

3. What is an event Listener?

----

3. What is the difference between an event handler and event Listener?
