Hall Ticket Number:

II/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

April, 2018 — **Common for CSE & IT**

Fourth Semester — **GUI Programming**

Time: Three Hours — **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1. Answer all questions (1X12=12 Marks)
   a) Write the syntax to create an object of a class.
   b) Write the uses of final keyword in java.
   c) What is an abstract class?
   d) What is an Exception?
   e) Define the term Thread.
   f) List any two collection classes of Java.
   g) What is stream?
   h) What is Applet? Write the syntax of <applet> tag.
   i) What is Event?
   j) What is Headless Exception?
   k) Differentiate awt and swings.
   l) List two constructors of JButton Class

UNIT I

2. a) Explain the following OOP principles in detail. Encapsulation, Inheritance, Polymorphism 6M
   b) Explain Declaration, Initialization and accesselements of Multidimensional Array with an example program. 6M

(OR)

3. a) Write a short note on Inheritance and types of Inheritance. 6M
   b) Explain the concept of Dynamic Method Dispatch with suitable example. 6M

UNIT II

4. a) Explain various string handling methods with suitable examples. 6M
   b) Explain StringBuffer Constructors and methods with an example. 6M

(OR)

5. a) Explain the concept of Interthread Communication in detail 6M
   b) Write a short on the concept of Thread priorities with suitable examples. 6M

UNIT III

6 a) Write a Java Program to copy contents of one file to another file using Character Streams. 6M
   b) Explain various Byte Streams in detail. 6M

(OR)

7. a) Explain the architecture of an Applet with its life cycle methods. 6M
   b) Explain various Event classes in detail. 6M

UNIT IV

8. a) Write a short note on creating a Swing Applet with suitable example. 6M
   b) Explain JTree class and create a list using suitable event handling functions in JTree. 6M

(OR)

9. a) Discuss various Layout Mangers to place the components in an Applet. 6M
   b) Explain Event handling using AWT components. 6M

II/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

April, 2018 — **Common for CSE & IT**

Fourth Semester — **GUI Programming**

Time: Three Hours — **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1. Answer all questions (1X12=12 Marks)

a) Write the syntax to create an object of a class.
ClassName obj = new ClassName();

b) Write the uses of final keyword in java.
i. To create constants.
ii. To prevent method overriding.
iii. To prevent inheritance.
} any two

c) What is an abstract class?
An abstract class is a class that is declared abstract —it may or may not include abstract methods. Abstract classes cannot be **instantiated**, but they can be **subclassed**.

d) What is an Exception?
An *exception* is an abnormal condition that arises in a code sequence at run time (or)
An **Exception** is a run-time error which interrupts the normal flow of program execution.

e) Define the term Thread.
a *thread* is a program's path of execution.

f) List any two collection classes of Java.
AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, LinkedList, ArrayList, ArrayDeque, AbstractSet, EnumSet, HashSet, LinkedHashSet, PriorityQueue, TreeSet, etc...
Note: Give full marks for any two classes

g) What is stream?
A *stream* is an abstraction that either produces or consumes information.

h) What is Applet? Write the syntax of <applet> tag.
applets are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document.
< APPLET
[CODEBASE = *codebaseURL*]
CODE = *appletFile*
[ALT = *alternateText*]
[NAME = *appletInstanceName*]
WIDTH = *pixels* HEIGHT = *pixels*
[ALIGN = *alignment*]
[VSPACE = *pixels*] [HSPACE = *pixels*]>
[< PARAM NAME = *AttributeName* VALUE = *AttributeValue*>]
[< PARAM NAME = *AttributeName2* VALUE = *AttributeValue*>]
. . .
[*HTML Displayed in the absence of Java*]
</APPLET>
} Any three

i) What is Event?
An *event* is an object that describes a state change in a source.

j) What is Headless Exception?

HeadlessException is thrown when code that is dependent on a keyboard, display, or mouse is called in an environment that does not support a keyboard, display, or mouse.

k) Differentiate awt and swings.

awt	swing
AWT components are platform-dependent.	Java swing components are platform-independent.
AWT components are heavyweight.	Swing components are lightweight.
AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT doesn't follows MVC(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC.

Note: Give full marks for any two differences.

l) List two constructors of JButton Class.

JButton()
JButton(**Icon** icon)
JButton(**String** text)
JButton(**String** text, **Icon** icon)

Note: Give full marks for any two constructors.

UNIT I

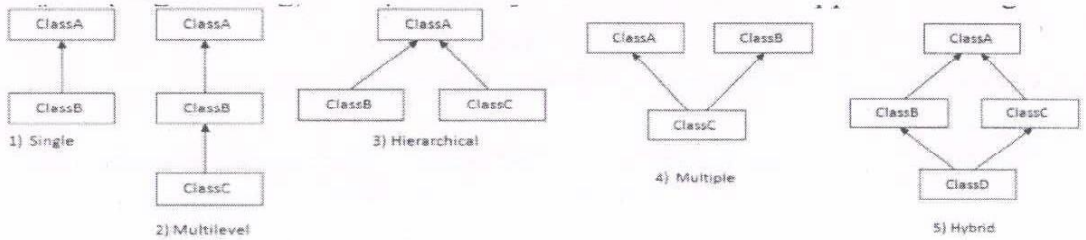
2. a) Explain the following OOP principles in detail.
Encapsulation, Inheritance, Polymorphism 6M

Encapsulation:

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**. } 2M

Inheritance:

Inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification.
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical. In java programming, multiple and hybrid inheritance is supported through interface only. } 2M



Polymorphism:

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

2M

There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism.

Compile time polymorphism can be achieved through method overloading.

Run time polymorphism can be achieved through method overriding.

b) Explain Declaration, Initialization and access elements of Multidimensional Array with an example program. 6M

multidimensional arrays are actually arrays of arrays.

Explanation: 4M

Example Program: 2M

Consider any valid Example Program

The syntax for declaring a two-dimensional array is:

data type []*arrayName*;

example: int[][] table;

The syntax to initialize a two-dimensional array.

```
Int[][] a new int[3][3];
for (int i = 0; i < a.length; i++) {
      for(int j = 0; j < a[i].length; j++) {
        a[i][j] = new Scanner(System.in).nextInt();
      }
}
```

(OR)

```
int[][] a = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
};
```

	Column1	Column2	Column3
Row1	1	2	3
Row2	4	5	6
Row3	7	8	9

The syntax to access elements of a two-dimensional array.

```
for (int i = 0; i < a.length; i++) {
      for(int j = 0; j < a[i].length; j++) {
        System.out.println(a[i][j]);
      }
}
```

(OR)

```
for(int i: a) {          //for-each loop
      System.out.println(i);
}
```

(OR)

3. a) Write a short note on Inheritance and types of Inheritance. 6M

Inheritance:

Inheritance is the process by which one object acquires the properties of another object.

This is important because it supports the concept of hierarchical classification.

On the basis of class, there can be three types of inheritance in java: single, multilevel and

hierarchical. In java, **multiple and hybrid inheritance is supported through interface only**.

Single Inheritance:

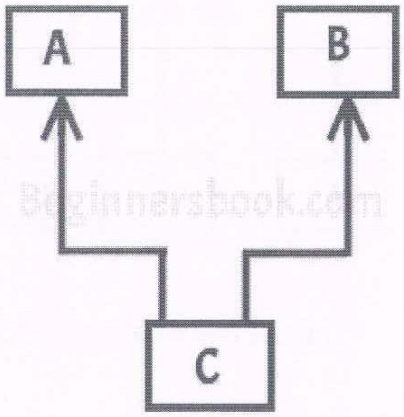
When a class extends another one class only then we call it a single inheritance.

Single Inheritance

Inheritance definition: 1M
Types of Inheritance: 5M
Note: Students may omit Multiple and hybrid inheritance because these two are not directly supported by Java

Multiple Inheritance:

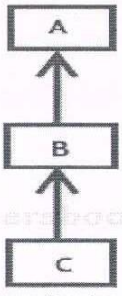
Multiple Inheritance refers to the concept of one class extending (Or inherits) more than one base class.



Multiple Inheritance

Multilevel inheritance:

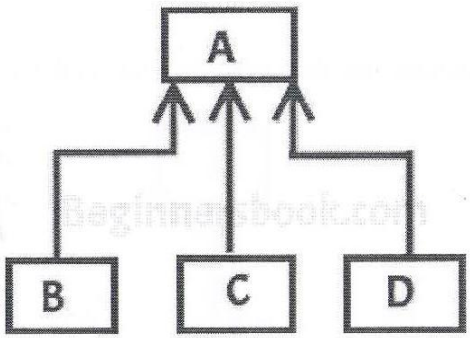
Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class.



Multilevel Inheritance

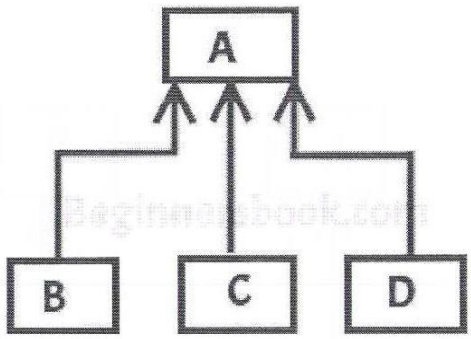
Hierarchical inheritance:

In Hierarchical inheritance one class is inherited by many sub classes.



Hierarchical Inheritance

Hybrid Inheritance:
In simple terms you can say that Hybrid inheritance is a combination of Single and Multiple inheritance.



Hierarchical Inheritance

b) Explain the concept of Dynamic Method Dispatch with suitable example. 6M

Method overriding forms the basis for one of Java's most powerful concepts: *dynamic method dispatch*. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. Dynamic method dispatch is important because this is how Java implements run-time polymorphism. } 2M.

```
// Dynamic Method Dispatch
class A {
        void callme() {
                System.out.println("Inside A's callme method");
        }
}
class B extends A {
        // override callme()
        void callme() {
                System.out.println("Inside B's callme method");
```

```
    }
}
class C extends A {
    // override callme()
    void callme() {
        System.out.println("Inside C's callme method");
    }
}
class Dispatch {
    public static void main(String args[]) {
        A a = new A(); // object of type A
        B b = new B(); // object of type B
        C c = new C(); // object of type C
        A r; // obtain a reference of type A
        r = a; // r refers to an A object
        r.callme(); // calls A's version of callme
        r = b; // r refers to a B object
        r.callme(); // calls B's version of callme
        r = c; // r refers to a C object
        r.callme(); // calls C's version of callme
    }
}
```

1

4M

Output:

```
Inside A's callme method
Inside B's callme method
Inside C's callme method
```

NOTE: Consider any valid example

This program creates one superclass called **A** and two subclasses of it, called **B** and **C**. Subclasses **B** and **C** override **callme( )** declared in **A**. Inside the **main( )** method, objects of type **A**, **B**, and **C** are declared. Also, a reference of type **A**, called **r**, is declared. The program then in turn assigns a reference to each type of object to **r** and uses that reference to invoke **callme( )**. As the output shows, the version of **callme( )** executed is determined by the type of object being referred to at the time of the call. Had it been determined by the type of the reference variable, **r**, you would see three calls to **A**'s **callme( )** method.

UNIT II

4. a) Explain various string handling methods with suitable examples. 6M

Method & Description
char charAt(int index) Returns the character at the specified index.
int compareTo(Object o) Compares this String to another Object.

int compareTo(String anotherString)
Compares two strings lexicographically.

int compareToIgnoreCase(String str)
Compares two strings lexicographically, ignoring case differences.

String concat(String str)
Concatenates the specified string to the end of this string.

boolean endsWith(String suffix)
Tests if this string ends with the specified suffix.

boolean equals(Object anObject)
Compares this string to the specified object.

boolean equalsIgnoreCase(String anotherString)
Compares this String to another String, ignoring case considerations.

byte getBytes()
Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

byte[] getBytes(String charsetName)
Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.

void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
Copies characters from this string into the destination character array.

int hashCode()
Returns a hash code for this string.

int indexOf(int ch)
Returns the index within this string of the first occurrence of the specified character.

int indexOf(int ch, int fromIndex)
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)
Returns the index within this string of the first occurrence of the specified substring.

int indexOf(String str, int fromIndex)
Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

int lastIndexOf(int ch)
Returns the index within this string of the last occurrence of the specified character.

int lastIndexOf(int ch, int fromIndex)

Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.

int lastIndexOf(String str)
Returns the index within this string of the rightmost occurrence of the specified substring.

int lastIndexOf(String str, int fromIndex)
Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

int length()
Returns the length of this string.

boolean matches(String regex)
Tells whether or not this string matches the given regular expression.

boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
Tests if two string regions are equal.

boolean regionMatches(int toffset, String other, int ooffset, int len)
Tests if two string regions are equal.

String replace(char oldChar, char newChar)
Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

boolean startsWith(String prefix)
Tests if this string starts with the specified prefix.

boolean startsWith(String prefix, int toffset)
Tests if this string starts with the specified prefix beginning a specified index.

String substring(int beginIndex)
Returns a new string that is a substring of this string.

String substring(int beginIndex, int endIndex)
Returns a new string that is a substring of this string.

char[] toCharArray()
Converts this string to a new character array.

String toLowerCase()
Converts all of the characters in this String to lower case using the rules of the default locale.

String toString()
This object (which is already a string!) is itself returned.

String toUpperCase() Converts all of the characters in this String to upper case using the rules of the default locale.
String trim() Returns a copy of the string, with leading and trailing whitespace omitted.
static String valueOf(primitive data type x) Returns the string representation of the passed data type argument.

Note: Give full marks for any ~~10~~ 6 (6x1) methods

Example:

```
class StringExample{
        public String toString(){
                return "StringExample";
        }
        public static void main(String[] r){
                StringExample se=new StringExample();
                String s1=new String("Java Class");
                char[] s2c={'2','C','S','E','C'};
                String s2=new String(s2c);


                System.out.println("Charactes @ index 1 in "+s1+" is:"+s1.charAt(1));
                char[] s1c=new char[s1.length()];
                s1.getChars(0,3,s1c,0);
                System.out.println("s1c contents are:");
                for(char c:s1c)
                        System.out.print(c);
                byte[] s1b=s1.getBytes();
                System.out.println();
                System.out.println("s1b contents are:");
                for(byte c:s1b)
                        System.out.print((char)c);
                System.out.println();
                System.out.println(s1+".equals("+s2+"):"+s1.equals(s2));
                String s3=s1;
                System.out.println("Java Class == Java Class:"+(s1.equals("Java Class")));


        System.out.println(s1+".regionmatches(5,"+s2+",1,1):"+s1.regionMatches(5,s2,1,1))
;
                System.out.println(s1+".comareTo("+s2+"):"+s1.compareTo(s2));

                System.out.println(s1+".IndexOf(\"a\") is:"+s1.indexOf("a"));
                System.out.println(s1+".lastIndexOf(\"a\") is:"+s1.lastIndexOf("a"));
```

```
        System.out.println(s1+".IndexOf('a',3) is:"+s1.indexOf('a',3));
        System.out.println(s1+".lastIndexOf('a',8) is:"+s1.lastIndexOf('a',6));

        System.out.println(s1+".concat("+s2+"):"+s1.concat(s2)+",s1="+s1);//s1+s2
        System.out.println(s1+".substring(1,5):"+s1.substring(1,5)+",s1="+s1);
        System.out.println(s1+".replace('a','@'):"+s1.replace('a','@')+",s1="+s1);
        s3="          "+s1+"    ";
        System.out.println("s3=@"+s3+"@"+s3+".trim():@"+s3.trim()+"@");
        System.out.println(s1+".toUpperCase():"+s1.toUpperCase()+",s1="+s1);

        s3=String.valueOf(10);
        //System.out.println(String.join(",",s1,s2,s3));
        System.out.println(se.toString());
    }
}
```

NOTE: Consider any valid example

b) Explain StringBuffer Constructors and methods with an example. 6M

Consructors:

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

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(char[] str, int offset, int len)

This method appends the string representation of a subarray 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(int i)
This method appends the string representation of the int argument to this sequence.

StringBuffer append(long lng)
This method appends the string representation of the long argument to this sequence.

StringBuffer append(Object obj)
This method appends the string representation of the Object argument.

StringBuffer append(String str)
This method appends the specified string to this character sequence.

StringBuffer append(StringBuffer sb)
This method appends the specified StringBuffer to this 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

void ensureCapacity(int minimumCapacity)
This method ensures that the capacity is at least equal to the specified minimum.

void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
This method characters are copied from this sequence into the destination character array dst.

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.

StringBuffer insert(int dstOffset, CharSequence s)
This method inserts the specified CharSequence into this sequence.

StringBuffer insert(int dstOffset, CharSequence s, int start, int end)
This method inserts a subsequence of the specified CharSequence into this sequence.

StringBuffer insert(int offset, double d)
This method inserts the string representation of the double argument into this sequence.

StringBuffer insert(int offset, float f)
This method inserts the string representation of the float argument into this sequence.

StringBuffer insert(int offset, int i
This method inserts the string representation of the second int argument into this sequence.

StringBuffer insert(int offset, long l)
This method inserts the string representation of the long argument into this sequence.

StringBuffer insert(int offset, Object obj)
This method inserts the string representation of the Object argument into this character sequence.

StringBuffer insert(int offset, String str)
This method inserts the string into this character 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.
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.

Note: Give full marks for any six methods

Example:

```
class StringBufferExample{
        public static void main(String[] r){
                StringBuffer sb=new StringBuffer();
                System.out.println("Length:"+sb.length());
                System.out.println("Capacity:"+sb.capacity());
                StringBuffer sb1=new StringBuffer(10);
                System.out.println("Length:"+sb1.length());
                System.out.println("Capacity:"+sb1.capacity());
                StringBuffer sb2=new StringBuffer("str1");
                System.out.println("Length:"+sb2.length());
                System.out.println("Capacity:"+sb2.capacity());
```

```
        //int ensureCapacity(int minCapacity)
        sb.ensureCapacity(20);
        System.out.println("Capacity:"+sb.capacity());

        sb2.setLength(2);
        System.out.println("Length:"+sb2.length());
        System.out.println("Capacity:"+sb2.capacity());

        //char charAt(int where)
        System.out.println("Character at 3:"+sb2.charAt(3));
        //void setCharAt(int where,char ch)
        System.out.print(sb2+"After setCharAt(3,'2'):");
        sb2.setCharAt(3,'2');
        System.out.println(sb2);
        //void getChars(int SI,int SE,char[] target, int TagetStart)
        char[] chars=new char[sb2.length()];
        sb2.getChars(0,4,chars,0);
        System.out.print("Chars Contents:");
        for(char ch:chars)
        System.out.print(ch);
        System.out.println();

        //StringBuffer append(String)
        //StringBuffer append(int)
        //StringBuffer append(Object)
        System.out.println(sb2+".append(10):"+sb2.append(10));

        //StringBuffer insert(int index,String str)
        //StringBuffer insert(int index,char ch)
        //StringBuffer insert(int index,Object o)
        System.out.println(sb2+".insert(2,\"ing\"):"+sb2.insert(3,"ing"));
        //StringBuffer reverse()
        System.out.println(sb2+".reverse():"+sb2.reverse()+",sb2="+sb2);
        //StringBuffer delete(int startIndex,int endIndex)
        //StringBuffer deleteCharAt(int startIndex)
        ////StringBuffer replace(int startIndex,int endIndex,String str)
        //string substring(int si)
        //string substring(int si,int ei)
    }
}
```

NOTE: Consider any valid example

(OR)

5. a) Explain the concept of Interthread Communication in detail. 6M

To avoid polling, Java includes an elegant interprocess communication mechanism via the **wait( )**, **notify( )**, and **notifyAll( )** methods. These methods are implemented as **final** methods in **Object**, so all classes have them. All three methods can be called only from within a **synchronized** context. Although conceptually advanced from a computer science

perspective, the rules for using these methods are actually quite simple:

- **wait( )** tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls **notify( )** or **notifyAll( )**.
- **notify( )** wakes up a thread that called **wait( )** on the same object.
- **notifyAll( )** wakes up all the threads that called **wait( )** on the same object. One of the threads will be granted access.

These methods are declared within **Object**, as shown here:

final void wait( ) throws InterruptedException
final void notify( )
final void notify All( )

Explanation : 3M
Program : 3M

Example:

```
class Buffer{
        int buffer[];
        boolean produced;
        Buffer(){
                buffer=new int[1];
        }
        void produce(int item){
                System.out.println(item+" Produced");
                buffer[0]=item;
        }
        int consume(){
                return buffer[0];
        }
}
class Producer extends Thread{
        Thread p;
        Buffer b;

        Producer(Buffer b){
                this.b=b;
                b.produced=false;
                p=new Thread(this,"Producer");
                p.start();
        }
        public void run(){
                int i=0;
                while(true){
                        synchronized(b){
                                while(b.produced){
                                        try{
                                                b.wait();
                                        }catch(Exception e){}
                                }
                                i++;
                                b.produce(i);
                                try{
```

```
                    Thread.sleep(1000);
                }catch(Exception e){}
                b.produced=true;

                b.notify();
            }
        }
    }
}
class Consumer extends Thread{
    Thread c;
    Buffer b;
    Consumer(Buffer b){
        this.b=b;
        c=new Thread(this,"Consumer");
        c.start();
    }
    public void run(){
        while(true){
            synchronized(b){
                while(!b.produced){
                    try{
                        b.wait();
                    }catch(Exception e){}
                }
                System.out.println(b.consume()+" Consumed");
                b.produced=false;
                b.notify();
            }

        }
    }
}
class PCDemo{
    public static void main(String[] args)throws Exception{
        Buffer b=new Buffer();
        Producer p=new Producer(b);
        Consumer c=new Consumer(b);
        p.p.join();
        c.c.join();
    }
}
```

NOTE: Consider any valid example

b) Write a short on the concept of Thread priorities with suitable examples. 6M

Thread priorities are used by the thread scheduler to decide when each thread should be allowed to run. In theory, over a given period of time, higher-priority threads get more CPU time than lower-priority threads. In practice, the amount of CPU time that a thread gets often depends on several factors besides its priority.

To set a thread's priority, use the **setPriority( )** method, which is a member of **Thread**. This is its general form:

final void setPriority(int *level*)

Here, *level* specifies the new priority setting for the calling thread. The value of *level* must be within the range **MIN_PRIORITY** and **MAX_PRIORITY**. Currently, these values are 1 and 10, respectively. To return a thread to default priority, specify **NORM_PRIORITY**, which is currently 5. These priorities are defined as **static final** variables within **Thread**.

You can obtain the current priority setting by calling the **getPriority( )** method of **Thread**, shown here:

final int getPriority( )

Explanation : 3M
Program : 3M

Example:

```
class TestMultiPriority1 extends Thread{
        public void run(){
                System.out.println("running thread name is:" +
                                                Thread.currentThread().getName());
                System.out.println("running thread priority is:"+
                                                 Thread.currentThread().getPriority());
        }
 public static void main(String args[]){
                TestMultiPriority1 m1=new TestMultiPriority1();
                TestMultiPriority1 m2=new TestMultiPriority1();
                m1.setPriority(Thread.MIN_PRIORITY);
                m2.setPriority(Thread.MAX_PRIORITY);
                m1.start();
                m2.start();
        }
}
```

Output:

```
running thread name is:Thread-0
running thread priority is:10
running thread name is:Thread-1
running thread priority is:1
```

UNIT III

6 a) Write a Java Program to copy contents of one file to another file using Character Streams. 6M

Program : 6M

```
import java.io.*;
public class FRFWFileCopy{
        public static void main(String args[]){
                try{
                        FileReader fr = new FileReader("Input.txt");
                        FileWriter fw = new FileWriter("Output.txt");
                        int k;
                        while( ( k = fr.read() ) != -1 ){
                                fw.write(k);
                                System.out.print((char) k);
                        }
                        fw.close();
                        fr.close();
```

```
            }
            catch(FileNotFoundException e){
                System.out.println("File does not exist. " + e);
            }
            catch(IOException e){
                System.out.println("Some I/O problem. " + e);
            }
        }
    }
```

one exception is enough

NOTE: Consider any valid example

b) Explain various Byte Streams in detail. 6M

Byte stream classes are used to perform reading and writing of 8-bit bytes. Streams being unidirectional in nature can transfer bytes in one direction only, that is, either reading data from the source into a program or writing data from a program to the destination. Therefore, Java further divides byte stream classes into two classes, namely, InputStream class and OutputStrearn class. The subclasses of InputStrearn class contain methods to support input and the subclasses of OutputStrearn class contain output related methods.

Input Stream Classes – 3M

Java's input stream classes are used to read 8-bit bytes from the stream. The InputStrearn class is the superclass for all byte-oriented input stream classes. All the methods of this class throw an IOException. Being an abstract class, the InputStrearn class cannot be instantiated hence, its subclasses are used. Some of these are listed in the following Table.

Class	Description
BufferedInputStream	contains methods to read bytes from the buffer (memory area)
ByteArrayInputStream	contains methods to read bytes from a byte array
DataInputStream	contains methods to read Java primitive data types
FileInputStream	contains methods to read bytes from a file
FilterInputStream	contains methods to read bytes from other input streams which it uses as its basic source of data
ObjectInputStream	contains methods to read objects
PipedInputStream	contains methods to read from a piped output stream. A piped input stream must be connected to a piped output stream
SequenceInputStream	contains methods to concatenate multiple input streams and then read from the combined stream

The Input Stream class defines various methods to perform reading operations on data of an input stream. Some of these methods along with their description are listed in the following Table.

Method	Description
int read()	returns the integral representation of the next available byte of input. It returns -1 when end of file is encountered
int read (byte buffer [])	attempts to read buffer. length bytes into the buffer and returns the total number of bytes successfully read. It returns -1 when end of file is encountered
int read (byte buffer [], int loc, int nBytes)	attempts to read 'nBytes' bytes into the buffer starting at buffer [loc] and returns the total number of bytes successfully read. It returns -1 when end of file is encountered

Note: Give full marks for four Stream classes from each category

int available ()	returns the number of bytes of the input available for reading
Void mark(int nBytes)	marks the current position in the input stream until 'nBytes' bytes are read
void reset ()	Resets the input pointer to the previously set mark
long skip (long nBytes)	skips 'nBytes' bytes of the input stream and returns the number of actually skippedbyte
void close ()	closes the input source. If an attempt is made to read even after closing the stream then it generates IOException

Output Stream classes

-3M

Java's output stream classes are used to write 8-bit bytes to a stream. The OutputStream class is the superclass for all byte-oriented output stream classes. All the methods of this class throw an IOException. Being an abstract class, the OutputStream class cannot be instantiated hence, its subclasses are used. Some of these are listed in the following Table.

Class	**Description**
BufferedOutputStream	Contains methods to write bytes into the buffer
ByteArrayOutputStream	Contains methods to write bytes into a byte array
DataOutputStream	Contains methods to write Java primitive data types
FileOutputStream	Contains methods to write bytes to a file
FilterOutputStream	Contains methods to write to other output streams
ObjectOutputStream	Contains methods to write objects
PipedOutputStream	Contains methods to write to a piped output stream
PrintStream	Contains methods to print Java primitive data types

The OutputStream class defines methods to perform writing operations. These methods are discussed in the following Table.

Method	**Description**
void write (int i)	writes a single byte to the output stream
void write (byte buffer [])	writes an array of bytes to the output stream
Void write(bytes buffer[],int loc, int nBytes)	writes 'nBytes' bytes to the output stream from the buffer b starting at buffer [loc]
void flush ()	Flushes the output stream and writes the waiting buffered output bytes
void close ()	closes the output stream. If an attempt is made to write even after closing the stream then it generates IOException

(OR)

7. a) Explain the architecture of an Applet with its life cycle methods. 6M

There are four java.Applet class methods that define the applet life cycle.

They are:

- **public void init()**: This method initializes the Applet and is invoked only once in the Applet life cycle. It helps to initialize variables and instantiate the objects and load the GUI of the applet. This is invoked when the page containing the applet is loaded. 1 ½
- **public void start():** This method is invoked after the init() method. It starts the execution of Applet. In this state, the applet becomes active. 1 ½
- **public void stop():** It stops the Applet execution. It's invoked when the Applet 1 ½

stops or when the browser is minimized. This makes the Applet temporarily inactive. The Applet frequently comes to this state in its life cycle and can go back to its start state.

- **public void destroy():** This destroys the Applet and is also invoked only once when the active browser page containing the applet is closed. 1½
- **public void paint() :** This method helps to create Applet's GUI such as a colored background, drawing and writing. It is a method of java.awt.Graphics package.

Example:(Optional)

```
import java.awt.Graphics;
import java.applet.Applet;
/*
<applet code="AppletEx.class" width="300" height="300">
</applet>
*/
public class AppletEx extends Applet{
        public void init(){
                System.out.println("Init");
        }
        public void start(){
                System.out.println("Start");
        }
        public void stop(){
                System.out.println("Stop");
        }
        public void destroy(){
                System.out.println("Destroy");
        }
        public void paint(Graphics g){
                System.out.println("Paint");
        }
}
```

NOTE: Consider any valid example

b) Explain various Event classes in detail. 6M

AWTEvent class defined in java.awt package is a subclass of EventObject. It is superclass (directly or indirectly) of all AWT-based events handled by delegation model.

Main Event Classes in java.awt.event.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list is double-clicked, or a menu item is selected. **Constructors:** ActionEvent(Object *src*, int *type*, String *cmd*) ActionEvent(Object *src*, int *type*, String *cmd*, int *modifiers*) ActionEvent(Object *src*, int *type*, String *cmd*, long *when*, int *modifiers*) **Methods:** String getActionCommand( ) int getModifiers( ) long getWhen( )

Note: Give full marks for any three Event classes

AdjustmentEvent	Generated when a scroll bar is manipulated. BLOCK_DECREMENT: The user clicked inside the scroll bar to decrease its value. BLOCK_INCREMENT: The user clicked inside the scroll bar to increase its value. TRACK The slider was dragged. UNIT_DECREMENT: The button at the end of the scroll bar was clicked to decrease its value. UNIT_INCREMENT: The button at the end of the scroll bar was clicked to increase its value. **Contructors:** AdjustmentEvent(Adjustable *src*, int *id*, int *type*, int *val*) **Methods:** Adjustable getAdjustable( ) int getAdjustmentType( ) int getValue( )
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible. COMPONENT_HIDDEN: The component was hidden. COMPONENT_MOVED: The component was moved. COMPONENT_RESIZED: The component was resized. COMPONENT_SHOWN: The component became visible. **Contructors:** ComponentEvent(Component *src*, int *type*) **Methods:** Component getComponent( )
ContainerEvent	Generated when a component is added to or removed from a container. **COMPONENT_ADDED** **COMPONENT_REMOVED** **Contructors:** ContainerEvent(Component *src*, int *type*, Component *comp*) **Methods:** Container getContainer( ) Component getChild( )
FocusEvent	Generated when a component gains or loses keyboard focus. **Contructors:** FocusEvent(Component *src*, int *type*) FocusEvent(Component *src*, int *type*, boolean *temporaryFlag*) FocusEvent(Component *src*, int *type*, boolean *temporaryFlag*, Component *other*) **Methods:** Component getOppositeComponent( ) boolean isTemporary( )
InputEvent	Abstract super class for all component input event classes. **Methods:**

	boolean isAltDown( ) boolean isAltGraphDown( ) boolean isControlDown( ) boolean isMetaDown( ) boolean isShiftDown( ) int getModifiers( ) int getModifiersEx( )
ItemEvent	Generated when a check box or a list item is clicked; also occurs when a choice selection is made or a checkable menu is selected or deselected. DESELECTED: The user deselected an item. SELECTED: The user selected an item. **Contructors:** ItemEvent(ItemSelectable *src*, int *type*, Object *entry*, int *state*) **Methods:** Object getItem( ) ItemSelectable getItemSelectable( ) int getStateChange( )
KeyEvent	Generated when input is received from the keyboard. **Contructors:** **KeyEvent** is a subclass of **InputEvent**. Here is one of its constructors: KeyEvent(Component *src*, int *type*, long *when*, int *modifiers*, int *code*, char *ch*) **Methods:** char getKeyChar( ) int getKeyCode( )
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component. MOUSE_CLICKED: The user clicked the mouse. MOUSE_DRAGGED: The user dragged the mouse. MOUSE_ENTERED: The mouse entered a component. MOUSE_EXITED: The mouse exited from a component. MOUSE_MOVED: The mouse moved. MOUSE_PRESSED: The mouse was pressed. MOUSE_RELEASED: The mouse was released. MOUSE_WHEEL: The mouse wheel was moved. **Contructors:** MouseEvent(Component *src*, int *type*, long *when*, int *modifiers*, int *x*, int *y*, int *clicks*, boolean *triggersPopup*) **Methods:** int getX( ) int getY( ) Point getPoint( ) int getClickCount( ) boolean isPopupTrigger( ) int getButton( )

TextEvent	Generated when the value of a textarea or textfield is changed. **Contructors:** TextEvent(Object *src*, int *type*)
WindowEvent	Generated when a window os activated, closed, deactivated, deiconified, iconified, opened, or quit. WINDOW_ACTIVATED: The window was activated. WINDOW_CLOSED: The window has been closed. WINDOW_CLOSING: The user requested that the window be closed. WINDOW_DEACTIVATED: The window was deactivated. WINDOW_DEICONIFIED: The window was deiconified. WINDOW_GAINED_FOCUS: The window gained input focus. WINDOW_ICONIFIED: The window was iconified. WINDOW_LOST_FOCUS: The window lost input focus. WINDOW_OPENED: The window was opened. WINDOW_STATE_CHANGED: The state of the window changed. **Contructors:** WindowEvent(Window *src*, int *type*, Window *other*) WindowEvent(Window *src*, int *type*, int *fromState*, int *toState*) WindowEvent(Window *src*, int *type*, Window *other*, int *fromState*, int *toState*) **Methods:** Window getWindow( ) Window getOppositeWindow( ) int getOldState( ) int getNewState( )

Note: Give full marks for any three event classes

UNIT IV

8. a) Write a short note on creating a Swing Applet with suitable example. 6M

An extended version of java.applet.Applet that adds support for the JFC/Swing component architecture.

Explanation : 1M
Program : 5M

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
//<applet code="TempConverter" height=300 width=100></applet>
public class TempConverter extends JApplet implements ActionListener{
        JTextField ctf,ftf;
        public void init(){
                setLayout(new FlowLayout());
                JLabel cl = new JLabel("Celsius");
                JLabel fl = new JLabel("Fahrenheit");

                ctf = new JTextField("0.0",20);
                ftf = new JTextField(20);
                ftf.setEditable(false);
```

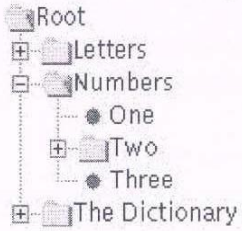
```
        JButton cbtn = new JButton("Convert");
        JButton rbtn = new JButton("Reset");

        add(cl);add(ctf);
        add(fl);add(ftf);
        add(cbtn);add(rbtn);
        cbtn.addActionListener(this);
        rbtn.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Convert")){
            if(!ctf.getText().equals("")){
                String cts=ctf.getText();
                double ct=Double.parseDouble(cts);
                double ft=(ct*1.8)+32;
                ftf.setText(""+ft);
            }
        }
        else if(ae.getActionCommand().equals("Reset")){
            ftf.setText("");
            ctf.setText("");
        }
    }
}
```

NOTE: Consider any valid example

b) Explain JTree class and create a JTree using suitable event handling functions. 6M

JTree:

With the JTree class, you can display hierarchical data. A JTree object does not actually contain data; it simply provides a view of the data. Like any non-trivial Swing component, the tree gets data by querying its data model. Here is a picture of a tree:



Explanation : 2M
Program : 4M

As the preceding figure shows, JTree displays its data vertically. Each row displayed by the tree contains exactly one item of data, which is called a *node*. Every tree has a *root* node from which all nodes descend. By default, the tree displays the root node, but you can decree otherwise. A node can either have children or not. We refer to nodes that can have children — whether or not they currently *have* children — as *branch* nodes. Nodes that cannot have children are *leaf* nodes.

Creating and Setting Up a Tree

Constructor or Method	Purpose
JTree()	Create a tree. The TreeNode argument specifies the root node, t

JTree(TreeNode) JTree(Object[]) JTree(Vector)	managed by the default tree model. The TreeModel argument specifies the model that provides the data to the table. The no-argument version of this constructor is for use in builders; it creates a tree that contains some sample data. If you specify a Hashtable, array of objects, or Vector as an argument, then the argument is treated as a list of nodes under the root node.
void setEditable(boolean)	The method sets whether the user can edit tree nodes. By default, tree nodes are not editable.
void setRootVisible(boolean)	Set whether the tree shows the root node.

Example:

```
import java.io.*;
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import java.awt.*;
class JTreeDemo extends JFrame implements TreeSelectionListener{
        JTree tree;
        JTextArea ja;
        JTreeDemo(){
                DefaultMutableTreeNode rn,jn,cn;
                rn = new DefaultMutableTreeNode("Root");
                jn = new DefaultMutableTreeNode("Java");
                jn.add(new DefaultMutableTreeNode("BankAccountDemo.java"));
                jn.add(new DefaultMutableTreeNode("Banner.java"));
                cn = new DefaultMutableTreeNode("C#");
                cn.add(new DefaultMutableTreeNode("BookStore.java"));

                rn.add(jn);rn.add(cn);
                tree = new JTree(rn);
                add(tree,BorderLayout.WEST);

                ja = new JTextArea(100,100);
                JScrollPane tasp = new JScrollPane(ja);
                add(tasp);

                pack();
                setVisible(true);
                tree.addTreeSelectionListener(this);
        }
        public void valueChanged(TreeSelectionEvent te){
                DefaultMutableTreeNode tn =
(DefaultMutableTreeNode)tree.getLastSelectedPathComponent();
                ja.setText("");
                try{
                        BufferedReader br=new BufferedReader(new
```

```
FileReader(tn.toString()));
                    String line="";
                    while((line=br.readLine())!=null){
                            ja.append(line+"\n");
                    }
                }catch(Exception e){}
        }
        public static void main(String[] args){
                new JTreeDemo();
        }
}
```

NOTE: Consider any valid example

(OR)

9. a) Discuss various Layout Mangers to place the components in an Applet. 6M

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout

Note! Give full marks for any three Layout Manages

Java BorderLayout

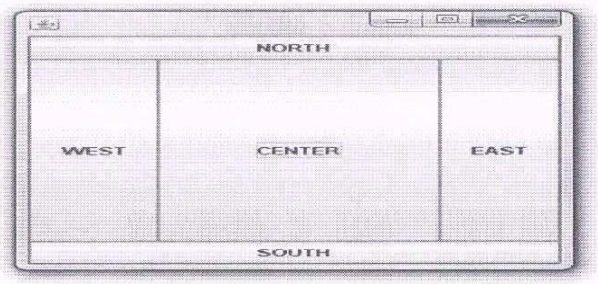
The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class:



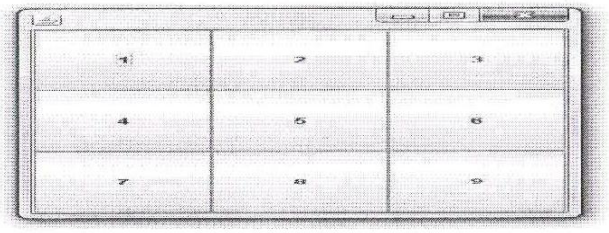
Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example of GridLayout class



Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

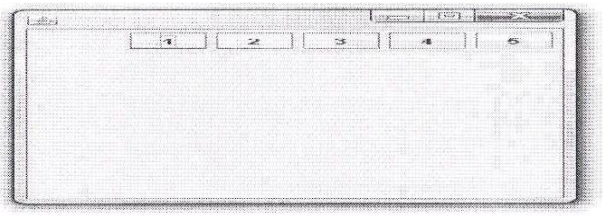
Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class



Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

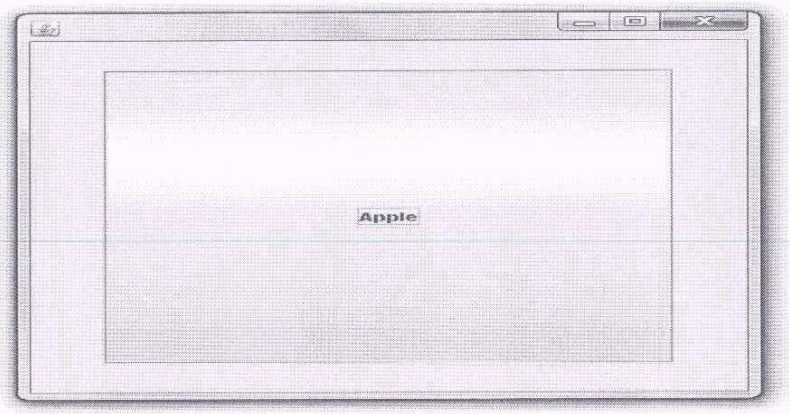
Commonly used methods of CardLayout class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the

given container.

- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Example of CardLayout class



Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

b) Explain Event handling using AWT components. 6M

Event handling using AWT components follows Event Delegation Model

The modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it receives an event. Once an event is received, the listener processes the event and then returns. The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to "delegate" the processing of an event to a separate piece of code.

In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them. This is a more efficient way to handle events than the design used by the original Java 1.0 approach. Previously, an event was propagated up the containment hierarchy until it was handled by a component. This required components to receive events that they did not process, and it wasted valuable time. The delegation event model eliminates this overhead.

6M

Note: underlined portion is optional

V. C

K. Hari Dep.

13/4/18