1) What do you mean by widening and narrow type conversions in Java? Give three examples for each.

Widening conversion: Type conversion is automatically done if the types are compatible and source type is smaller than destination type. 0.5M

Example: a) double d=20; // integer to double 1.5M

b) int i=200;

long l=i; // integer to long

c) float f=23.5;

double d= f; // float to double

Narrowing conversion: But when destination type is smaller than source type, then we need to explicitly cast it. This is also called narrowing conversion. 0.5M

```
byte b;
int i = 81;                                          1.5M
double d = 323.142;
float f = 72.38f;
char c = 'A';

a)  c = (char) i;
System.out.println("i = " + i + " c = " + c);

b)  i = (int) d;
System.out.println("d = " + d + " i = " + i); // LINE B

c)  i = (int) f;
System.out.println("f = " + f + " i = " + i); // LINE D

b = (byte) d;
System.out.println("d = " + d + " b = " + b);
```

2) Explain the two short circuit logical operators available in java with examples.

Logical AND (&&):

Logical AND performs a Boolean AND operation on its operands. It evaluates to true if and only if both operands of logical AND are true. If either or both operands are false, it evaluates to false.

Logical AND or conditional AND operator is called short-circuit operator because it conditionally evaluates its second operand. If the first operand evaluates to false, the value of the expression is false, regardless of the value of the second operand. Therefore, to increase efficiency, the Java interpreter takes a short-cut and skips the second operand. Since the second operand is not guaranteed to be evaluated, you must use caution when using this operator with expressions that have side effects. On the other hand, the conditional nature of this operator allows us to write Java expressions such as the following: 1.5M

```
if (data != null && i < data.length && data[i] != -1) ...   0.5M
```

Logical OR (||) :

Logical OR operator performs a boolean OR operation on its two boolean operands. It evaluates to true if either or both of its operands are true. If both operands are false, it evaluates to false. Like the && operator, || does not always evaluate its second operand. If the first operand evaluates to true, the value of the expression is true, regardless of the value of the second operand. Thus, the operator simply skips that second operand in that case. 1.5M

Example:

```
If ( yr %400 ==0|| (yr%4==0  && yr%100!=0)
      System.out.println( yr + "is a leap year");   0.5M
```

3) Write a Java program to define a class called Book with the private data members title, author and numOfPages;

Define zero argument constructor and a parameterized constructor and a display member function.

Write a separate class called BookDemo to define main method. The main method should create an array of n Book objects. The main () also reads an author name from keyboard and displays all the books' Title and Author name which matches with the author name read.

```
public class Book                2M
{       private String title, author;
        private int numOfPages;
                                    /* zero argument constructor */
        public Book ()
        {
           title = null;
           author = null;
           numOfPages = 0;
        }
        /* constructor with parameters */
        Book (String title, String author,    int numOfPages)
        {
           this.title = title;
           this.author = author;
           this.numOfPages = numOfPages;
        }
        String getAuthor()
        {
           return author;
        }
       void display()
       { System.out, println( " "+ title +"  " +author+ numOfPages);
       }
}
```

```
class BookDemo                  2M
{
  public static void main(String args[])
  { Scanner sc=new Scanner(System.in);
    System.out.println("Enter the number of books");
    Book bks[]=new Book[sc.nextInt()];
    String title, author; int pages ;
    for ( int i=0;i<10;i++)
    {
      title= sc.nextLine();
      author=sc.nextLine();
      pages=sc.nextInt();
      bks[i] = new Book(title,author,pages);
    }

System.out.println("Enter the name of the author whose books you want to
see");

String author1=sc.nextLine();

for ( int i=0;i<10;i++)
    {  if (author1.equals(bks[i].getAuthor())
         bks[i].display();
    }
   }
}
```

4 a) Explain the use of 'this' keyword in java with an example. 2M

this is a **keyword in Java**. Which can be **used** inside method or constructor of a class. It(this) works as a reference to current object whose method or constructor is being invoked. this **keyword** can be **used** to refer any member of current object from within an instance method or a constructor.

1M

```
class Box                                                    1M
{
int l,b,h;
Box(int l,int b,int h)
{
    this.l=l;
    this.b=b;
     this.h=h;
}

}
```

b) How do you create a 2D array in Java which has 3 rows and the no. of columns in 1st row is 3 , 2nd is 5 and 3rd is 2. 2M

```
int a[][]=new int[3][];
  a[0]=new int[3];
  a[1]=new int[5];
  a[2]=new int[2];
```

5. a) What do you mean by dynamic method dispatch. Explain the concept of dynamic method dispatch with the help of a java program. 3M

- Ans:

This is the mechanism by which a **call to an overridden method is resolved at run-time**, rather than compile-time. Through this , java implements run-time polymorphism.

- When an overridden method is called through a super class reference, java determines which version of that method to execute based

upon the type of the object being referred to at the time the call occurs. This is made at run-time. (1M)

```
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"); (1M)
                }   }
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
                }
        }(1M)
```

b) Mention the order of invocation of constructor in multilevel hierarchy. (1M)

Ans:

- In a class hierarchy, constructors are called in order of derivation, from superclass to subclass (0.5M)
- Since **super( )** must be the first statement executed in a subclass' constructor, this order is the same whether or not **super( )** is used
- If **super( )** is not used, then the default or parameter-less constructor of each superclass will be executed (0.5M)

6.a) With suitable examples distinguish between method overloading and method overriding.(2)

Ans:

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass(0.5M)

```
class A {
        int i, j;
        A(int a, int b) {
                i = a;
                j = b;      }
        void show() {
                System.out.println("i and j: " + i + " " + j);
                }     }
class B extends A {
        int k;
        B(int a, int b, int c) {
                super(a, b);
                k = c;      }
        // display k – this overrides show() in A
```

```
    void show() {
        System.out.println("k: " + k);
        }
    }(0.5M)
// Methods with differing type signatures are overloaded – not overridden.(0.5M)
class A {
    int i, j;
    A(int a, int b) {
        i = a;
        j = b;   }
    void show() {
        System.out.println("i and j: " + i + " " + j);
        }       }       // Create a subclass by extending class A.
class B extends A {
    int k;
    B(int a, int b, int c) {
        super(a, b);
        k = c;    }          // overload show()
    void show(String msg) {
        System.out.println(msg + k);
        }    }(0.5M)
```

b) With examples mention any two uses of super keyword. (2)

Ans:

super has two general forms: (i) calls the superclass' constructor,

(ii) The second is used to access a member of the superclass that has been hidden by a member of a subclass (1M)

```
class BoxWeight extends Box {
        double weight;          // weight of box
        // initialize width, height, and depth using super()
        BoxWeight(double w, double h, double d, double m) {
            super(w, h, d);       // call superclass constructor
            weight = m;
            }
    }(0.5M)
class A {
    int  i;    }
    // Create a subclass by extending class A.
class B extends A {
    int i; // this i hides the i in A
    B(int a, int b) {
        super.i = a;    // i in A
        i = b;          // i in B
        }  (0.5M)
```

7. a) What is an abstract class? Why do we need abstract classes? (2M)

Ans:

It may be needed to define a superclass that declares the structure of a given abstraction without giving a complete implementation of every method. Such a class determines the nature of the methods that the subclass must implement. (1M)

This is - when a superclass is unable to create a meaningful implementation for a method.

To ensure that a subclass overrides all necessary methods – java provides "abstract method" (1M)

b) With the help of an example , demonstrate overloading of constructors. (2M)

Ans:

A constructor initializes an object immediately upon creation.It has the same name as the class in which it resides and is syntactically similar to a method. Has no return type, not even void. There can be parameter less constructors and parameterized constructors. If constructors are not written, then default constructor is invoked. 1M

```
class Box {
        double width;
        double height;
        double depth;
                // constructor used when all dimensions specified
        Box(double w, double h, double d) {
                width = w;
                height = h;
                depth = d;
                }
        // constructor used when no dimensions specified
        Box() {
                width = -1; // use -1 to indicate
                height = -1; // an uninitialized
                depth = -1; // box
                }
        1M
        // constructor used when cube is created
        Box(double len) {
                width = height = depth = len;
                }               // compute and return volume
        double volume() {
                return width * height * depth;
                }
         }
```

8. a)Show an example of overloaded varargs method that is ambiguous. 2M
Ans:

Example:

```
static void vaTest(int ... v) {                    }
static void vaTest(int, int ... v) {               }
```

- Suppose the function call is:

```
vaTest(1);// ambiguous                                    2M
```

b) How are objects passed as parameters to methods in Java? Explain with the help of an example. (2M)

Ans:

```
class Test {
int a, b;
Test(int i, int j) {
a = i;
b = j;
}
boolean equals(Test o) {
if(o.a == a && o.b == b) return true;
else return false;
}                                                                    1M
}
class PassOb {
public static void main(String args[]) {
Test ob1 = new Test(100, 22);
Test ob2 = new Test(100, 22);
Test ob3 = new Test(-1, -1);
System.out.println("ob1 == ob2: " + ob1.equals(ob2));  1 marks
System.out.println("ob1 == ob3: " + ob1.equals(ob3));
}
}                                                                    0.5M
```

As you can see, the equals( ) method inside Test compares two objects for equality and returns the result. That is, it compares the invoking object with the one that it is passed. If they contain the same values, then the method returns true. Otherwise, it returns false. Notice that the parameter o in equals( ) specifies Test as its type. Although Test is a class type created by the program, it is used in just the same way as Java's built-in types. 0.5M