Internal Assessment Test 1 November 2024									
Sub:	**Programming in Java**				Sub Code:	15CS561	Branch:	TCE/ECE	
Date :	15 -10 -19	Duration:90 m	Max Marks:	50	Sem /Sec:	IIIA B/C		OBE	

	Answer any FIVE FULL Questions	Marks	CO	RBT
1 (a)	**Distinguish between method overloading and method overriding. Write Java programs to demonstrate the use of method overloading and method overriding.** To implement this concept, the constraints are: • The number of arguments should be different, and/or • Type of the arguments must be different. class Overload { void test() //method without any arguments { System.out.println("No parameters"); } void test(int a) //method with one integer argument { System.out.println("Integer a: " + a); } void test(int a, int b) //two arguments { System.out.println("With two arguments : " + a + " " + b); } void test(double a) //one argument of double type { System.out.println("double a: " + a); } } class OverloadDemo { public static void main(String args[]) { Overload ob = new Overload(); ob.test(); ob.test(10); ob.test(10, 20); ob.test(123.25); }	10	CO3	L1, L4

	} • In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its super class, then the method in the subclass is said to ***override*** the method in the super class. • When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the super class will be hidden. class A { int i, j; A(int a, int b) { i = a; j = b; } **void show() //suppressed** { 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; } **void show() //Overridden method** { System.out.println("k: " + k); } } class Override { public static void main(String args[]) { B subOb = new B(1, 2, 3); **subOb.show();** } }			
2. (a)	**List and explain the uses of the keyword 'super' with Java programs.** • Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super. • super has two general forms. • The first calls the superclass' constructor.	6	CO3	L2

- The second is used to access a member of the superclass that has been hidden by a member of a subclass.

Using super to Call Superclass Constructors

A subclass can call a constructor defined by its superclass by use of the following form of super:

super(arg-list);

- Here, arg-list specifies any arguments needed by the constructor in the superclass.
- constructor.

Example

```
class Box {
private double width;
private double height;
private double depth;
// construct clone of an object
Box(Box ob) { // pass object to constructor
width = ob.width;
height = ob.height;
depth = ob.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
}
// compute and return volume
double volume() {
return width * height * depth;
}
}
// BoxWeight now fully implements all constructors.
class BoxWeight extends Box {
double weight; // weight of box
// constructor when all parameters are specified
BoxWeight(double w, double h, double d, double m) {
super(w, h, d); // call superclass constructor
weight = m;
}
// default constructor
BoxWeight() {
super();
weight = -1;
}
}
```

class DemoSuper {
public static void main(String args[]) {
BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
BoxWeight mybox3 = new BoxWeight(); // default
double vol;
vol = mybox1.volume();
System.out.println("Volume of mybox1 is " + vol);
System.out.println("Weight of mybox1 is " + mybox1.weight);
System.out.println();
vol = mybox3.volume();
System.out.println("Volume of mybox3 is " + vol);
System.out.println("Weight of mybox3 is " + mybox3.weight);
System.out.println();
}
}

Second use of super to access a member of the superclass:

- *The second form of super always refers to the superclass of the subclass in which it is used.*

The general form is
super.member

- *Here, member can be either a method or an instance variable. The second form of super is most*

applicable to situations in which member names of a subclass hide members by the same name in
the superclass.
Example:

/ Using super to overcome name hiding.

- *Here, member can be either a method or an instance variable. The second form of super is most*

applicable to situations in which member names of a subclass hide members by the same name in
the superclass.
Example:

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
}
void show() {
System.out.println("i in superclass: " + super.i);
System.out.println("i in subclass: " + i);
}
}
class UseSuper {

	public static void main(String args[]) { *B subOb = new B(1, 2);* *subOb.show();* *}* *}* *This program displays the following:* *i in superclass: 1* *i in subclass: 2*			
3	What is constructor? Name and explain different types of constructor with example program. Constructor is a special type of member method which is invoked automatically when the object gets created. Constructors are used for object initialization. They have the same name as that of the class. Since they are called automatically, there is no return type for them. Constructors may or may not take parameters ● Every class is provided with a ***default constructor*** which initializes all the data members to respective *default values*. (Default for numeric types is zero, for character and strings it is null and default value for Boolean type is false.) ● In the statement *classname ob= new classname();* the term *classname()* is actually a constructor call. ● If the programmer does not provide any constructor of his own, then the above statement will call default constructor. ● If the programmer defines any constructor, then default constructor of Java cannot be used. ● So, if the programmer defines any parameterized constructor and later would like to create an object without explicit initialization, he has to provide the default constructor by his own. For example, the above program, if we remove ordinary constructor, the statements like Box b1=new Box(); will generate error. To avoid the error, we should write a default constructor like – Box(){ } Now, all the data members will be set to their respective default values. . class Box { double w, h, d; double volume() { return w*h*d; } Box() //**ordinary constructor** { w=h=d=5; } Box(double wd, double ht, double dp) //**parameterized constructor** { w=wd; h=ht; d=dp;	4	CO3	L3

	} }			
4a	What is Type casting? Illustrate with an example, what is meant by automatic type promotion. ● When we assign a value of one type to a variable of another type, if the two types are compatible, then Java will perform the conversion automatically. Example assigning an int value to a long variable. ● If the two types are not compatible, then Java will not perform the implicit conversion. Example while assigning a double value to a byte variable no automatic conversion will happen. b = (byte) a; However, it is still possible to obtain a conversion between incompatible types by using casting. Casting performs an explicit conversion between incompatible types. **Java's Automatic Conversions:** ● When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met: ➢ The two types are compatible. ➢ The destination type is larger than the source type ● When these two conditions are met, a widening conversion takes place. ● For widening conversions, the numeric types, including integer and floating-point types are compatible with each other. ● There is no automatic conversions from the numeric types to char or boolean. Also char or boolean are not compatible with each other. ● Java also performs an automatic type conversion when storing a literal integer constant into variables of type byte, short, long or char. **Casting Incompatible Types:** ● If we want to assign an int value to a byte variable, conversion will not be performed automatically, because a byte is smaller than an int. ● This kind of conversion is called narrowing conversion since we are explicitly making the value narrower so that it will fit into the target type. ● To create a conversion between the two incompatible types, we must use a cast. A cast is simply an explicit type conversion. ● The general form of cast is - (target-type) value target-type specifies the desired type to convert the specified value to. For example to cast an int to a byte int a=20; byte b; b= (byte) a; ● If the integer value is larger than the range of a byte, it will be reduced modulo (the remainder of an integer division by the) byte's range. ● A different type of conversion called truncation will occur when a floating-point value is assigned to an integer type. Integers do not have fractional components.	10	CO3	L1, L3

Hence when a floating point value is assigned to an integer type, the fractional component is lost.

● For example, if the value 1.23 is assigned to an integer, the resulting value will be 1. The 0.23 will be truncated.

● If the size of the whole number component is too large to fit into the target integer type, then the value will be reduced modulo the target type's range.

3b. How to declare two dimensional arrays in java ? explain with an simple example.

● Syntax:

```
type var-name[ ] [ ];
var-name = new type [size] [size];
or
type var-name [ ] [ ] = new type [size] [size];
```

example

```
class TwoDArray
{
public static void main(String args[])
{
int twoD [ ] [ ] = new int [4] [5];
int i,j,k=0;
for(i=0;i<4;i++)
for(j=0;j<5;j++)
{
twoD[i][j] = k;
k++;
}
for(i=0;i<4;i++)
{
for(j=0;j<5;j++)
System.out.print (twoD[i][j] + " ");
System.out.println();
}
}
}
```

4b

```
public class BOX {
        public static void main(String[] args) {
                BOX b1 = new BOX();
                BOX b2=b1;
                System.out.println(b1);
                b1=2;
                System.out.println(b1);
        }
}
```

[10] CO3 L1, L3

Does above java code get compile? If provide output write the reason., if no write the reason and correct answer.

No. Integer cannot be assigned to b1.

5a	With an example explain finalize() method in java • Java provides a mechanism called finalization to handle situations where specific actions are required before an object is reclaimed by the garbage collector. • Objects may hold non-Java resources like file handles or character fonts. • It is essential to free these resources before an object is destroyed. • To add a finalizer to a class, define the finalize() method. • The Java runtime automatically calls this method when it is about to recycle an object of that class. • • Inside the finalize() method, specify actions that must be performed before an object is destroyed. • This method is called by the garbage collector just before reclaiming the object • The garbage collector runs periodically to identify and reclaim objects that are no longer referenced. • Objects without any live references are candidates for garbage collection. • Just before an object is freed, the Java runtime invokes the finalize() method on that object. • This provides an opportunity to release resources or perform other necessary cleanup tasks. • The garbage collector identifies and marks objects that are eligible for finalization. • Finalization ensures proper resource management and cleanup before the object is deallocated. The **finalize( ) method has this general form:** protected void finalize( ) { // finalization code here }	[10]	CO3	L2
5a	Write a note on use of 'this' keyword • Sometimes a method will need to refer to the object that invoked it. • **this** can be used inside any method to refer to the *current object. That is,* **this** is always a reference to the object on which the method was invoked. You can use **this** anywhere a reference to an object of the current class ' type is permitted. **Example** **// A redundant use of this** Box(double w, double h, double d) { this.width = w; this.height = h; this.depth = d; }	[10]	CO3	L4

6a Discuss Lexical issues in JAVA program

- **Whitespace:** Java is a free-form language. In Java, whitespace is a space, tab, or newline.
- **Identifiers:** Used for class names, method names, and variable names. An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters. They must not begin with a number, again, Java is case- sensitive.
  - AvgTemp count, a4, $test, this_is_ok are Valid
  - 2count, high-temp, Not/ok are Invalid
- **Literals:** A constant value in Java is created by using a literal representation of it. It can be used anywhere a value of its type is allowed.
- 100, 98.6, 'X', "This is a test"

- **Comments:** As there are three types of comments defined by Java.

1. Single comment
2. Multiline
3. documentation comment

Documentation comment is used to produce an HTML file that documents your program.
The documentation comment begins with a /** and ends with a */.

- **Separators** : The most commonly used separator in Java is the semicolon. As you have seen, it is used to terminate statements.

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

Java Keywords

There are 50 keywords currently defined in the Java language.
These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language.
These keywords cannot be used as names for a variable, class, or method.
The keywords const and goto are reserved but not used.
in addition to the keywords, Java reserves the following: true, false, and null.
These are values defined by Java

		abstract	continue	for	new	switch			
		assert	default	goto	package	synchronized			
		boolean	do	if	private	this			
		break	double	implements	protected	throw			
		byte	else	import	public	throws			
		case	enum	instanceof	return	transient			
		catch	extends	int	short	try			
		char	final	interface	static	void			
		class	finally	long	strictfp	volatile			
		const	float	native	super	while			
6. (b)	final abstract class A{ B(); } Public Class B extends A{ B(){ System.out.println(“Test”); } } Public class C{ public static void main(String[] args){ A a1 = new B(); } } Does above java code get compile? If provide output write the reason., if no write the reason and correct answer. No. final and abstract cannot be used together for a class.						[06]	CO3	L2