Subject Name: Object Oriented Programming using JAVA - QUESTION BANK SOLUTION
Subject Code: 2150704 Unit-1 And 2

Faculties: MS. HEMALI MOJIDRA (SHAH)

Sr No	QUESTIONS	Marks
	UNIT-1 BASICS OF JAVA:	
	TOPIC:1 (About Basic Java) Features of Java, Byte Code and Java Virtual Machine, JDK	
1	JVM is platform dependent. Justify**. (May-13) [LJIET]** **OR** Justify statement. (i)JVM is platform dependent **(Dec-15) [LJIET]** **Ans:** JVMs are available for many hardware and software platforms. **JVM, JRE and JDK are platform dependent** because configuration of each OS differs. The **JVM executes Java code**. **JVM is written in platform specific languages** such as **C/C++/ASM** etc. The JVM is not written in Java and hence cannot be platform independent. **Require to build different JVM for different Platform(Operating System).** JVM **depends on the operating system**. JVMs are **not** platform **independent**. In fact they are platform specific run time environment provided by the vendor**. Each platform (Windows, UNIX, Mac etc) has its own JVM** to run Java applications.	**3/4**
2	(ii)There is no destructor in Java. **(May-13,Dec-15) [LJIET] .** **Ans: True-** Java has its own automatic memory de- allocation of memory management mechanism - garbage collection- (GC) implementation. so it does not require any destructor like C++ to deallocate memory . Garbage collector identifies the objects which no longer needed in program and free the memory occupied by object. **finalize() method of Object class is invoked when garbage collector is invoked before deleting object of class.** Developer need **to override finalize() method** in class **to release the resources** occupied by the class object. Implementation of this **finalize() method work as the destructor.** So there is no destructor in java.	**3/4**
3	(i) Java program is to be compiled first and then to be interpreted for execution. True or false? Justify your answer. **(Dec-15) [LJIET]** **Ans: TRUE**- Java source file **.java** compiled using **javac** command and generates the **bytecode** (.class) file. This **.class** file contains the **bytecode** which is **interpreted** by the **JVM**(Java Virtual Machine) than execute it and generate the output.	**3/4**
1	Explain features of JAVA. **(June-12, Dec-13, June-14) [LJIET] OR** List various features of Java? Also explain any two feature with example. **(May-15,May-16) [LJIET]** **Ans: OR java buzzwords** There is given many features of java. They are also known as **java buzzwords**. The Java Features given below are simple and easy to understand. 1. Simple 2. Object-Oriented 3. Platform independent 4. Secured	**7**

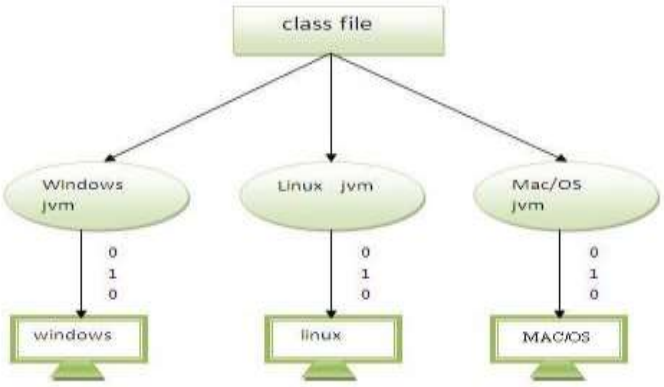
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted but High Performance
10. Multithreaded
11. Distributed

1 Simple : According to Sun, Java language is simple because:
- syntax is based on C++ (so easier for programmers to learn it after C++).
-removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.
- No need to remove(free memory) unreferenced(unused) objects because there is automatic Garbage Collection in java.

2 Object-oriented: Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior. Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules. Basic concepts of OOPs are:
Object
Class
Inheritance
Polymorphism
Abstraction
Encapsulation

3 Platform Independent: A **platform** is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms. It has two components: 1) Runtime Environment 2) API(Application Programming Interface)

java is platform independent Java code can be run on multiple platforms e.g.Windows,Linux,Sun Solaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This **bytecode is a platform independent** code because it can be run on multiple platforms i.e. Write Once and Run Any where(WORA).



4 Secured: Java is secured because: No explicit pointer And Programs run inside virtual machine sandbox.
Classloader- adds security by separating the package for the classes of the local file system from those that are imported from network sources.

Bytecode Verifier- checks the code fragments for illegal code that can violate access right to objects.

Security Manager- determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL, JAAS, cryptography etc.

5 **Robust:** Robust simply means strong. Java uses **strong memory management**. There are lack of pointers that avoids security problem. There is **automatic garbage collection** in java. **There is exception handling and type checking mechanism** in java. All these points makes java robust.

6 **Architecture-neutral:** There is no implementation dependent features e.g. size of primitive types is set. Any java develoed program always work though OS upgrade and JVM upgrade. Java guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. Though Operating system upgrades, processor upgrades, and changes in core system resources , java program always - "write once; run anywhere, any time, forever."

7 **Portable:** We may carry the java bytecode to any platform. Bytecode is platform independent and executed by JVM.

8 **Interpreted but High Performance:** Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++). **Just-In-Time(JIT) compiler:** It is used to improve the performance of java program . JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU which executes fast.

9 **Distributed:** We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

10 **Multi-threaded:** A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

2 Describe the following features of java: 1)Multithreaded 2)Architecture-neutral 3)Interpreted 4)High performance 5)Distributed 6)Portable 7) Dynamic **(Dec-14) [LJIET]** — **7**

Ans: Same as Answer of Q-1 of Topic-1

3 Compare Object oriented programming with sequential programming. **(May-15) [LJIET].** — **7**

	Procedure Oriented/Sequencial Programming	**Object Oriented Programming**
Divided Into	In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
Importance	In POP,Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from	In OOP, objects can move and

		function to function in the system.	communicate with each other through member functions.
	Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
	Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
	Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
	Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
	Inheritance	Not Support Inheritance.	OOP support inheritance – extending existing property of class.
	Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

4 List OOP characteristics and describe inheritance with examples. **(May-13) [LJIET] .** **7**

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object: Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class **Collection of objects** is called class. It is a logical entity.

Abstraction: Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

Encapsulation: Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Inheritance: When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism: When **one task is performed by different ways** i.e. known as

polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

- **Abstraction** – Hiding the access of properties (class keyword)
- **Encapsulation** – Binding data and method which operates on data in single body- Two parts of capsule – one is data and other is Methods and body work as a binding (Keywords – class, interface)
- **Inheritance** – Hiring the properties of existing class (Keywords – extends, implements)
- **Polymorphism** – Single thing is available in many forms (Poly –many, morphism – forma)
    - **Types Of Polymorphism**

<table>
<tr><td>- Compile Time</td><td>- Runtime</td></tr>
<tr><td>Over loading</td><td>Over Ridding</td></tr>
<tr><td>Static binding</td><td>Dynamic Binding</td></tr>
<tr><td>Early Binding</td><td>Late Binding</td></tr>
<tr><td>Same thing with differ parameter.</td><td>Same method signature in parent and child class</td></tr>
<tr><td>- Types Of Over Loading
<table>
<tr><td>1. Method/Function</td><td>2. Constructor</td></tr>
<tr><td>class A
{
void msg(){}
void msg(int i){}
}</td><td>class A()
{
A(){}
A(int i){}
}</td></tr>
<tr><td>same method with different parameter</td><td>constructor with different parameter</td></tr>
</table></td><td>- No Types
<table>
<tr><td>class A
{
void msg(){}
void msg(int i){}
}</td></tr>
<tr><td>class B extends A
{
void msg(){
}
void msg(int i){
}
}</td></tr>
<tr><td>same method with same parameter in s</td></tr>
</table></td></tr>
</table>

Inheritance : can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields

of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

Why use inheritance in java

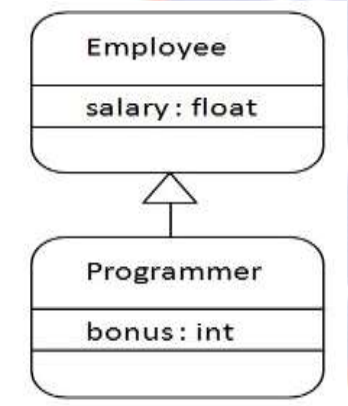
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

extends Keyword :
extends is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.

```
class Super{ ...
}

class Sub extends Super{ ..
}
```

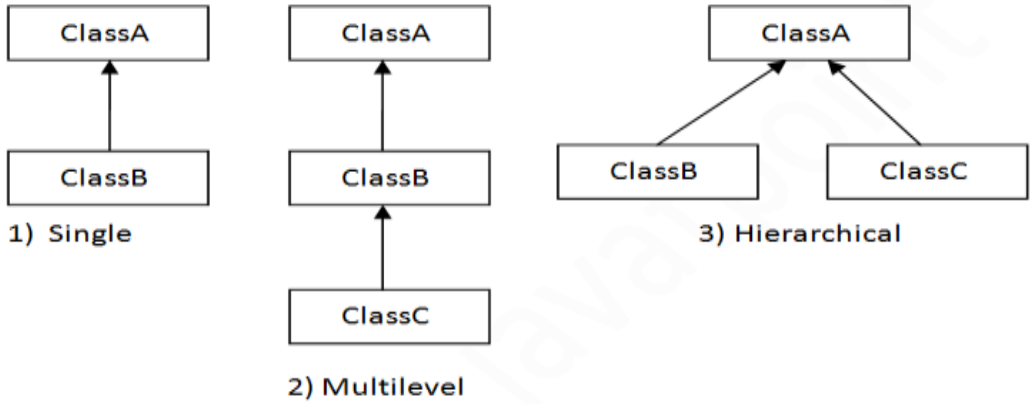
Understanding the simple example of inheritance:



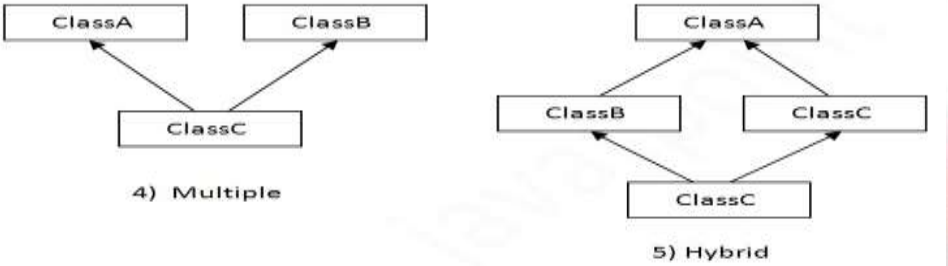
As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes is **Programmer IS-A Employee**.It means that Programmer is a type of Employee.

Types of inheritance in java

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. When a class extends multiple classes i.e. known as multiple inheritance. For Example:



Why multiple inheritance is not supported in java using class?:

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

```
class A{
      void msg(){System.out.println("Hello");}
}
class B{
      void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

       Public Static void main(String args[]){
         C obj=new C();
         obj.msg();//Now which msg() method would be invoked?
       }
}  Output: Compile Time Error
```

TOPIC:2(Data types, Operator, Control Statement)

Data types, Operator Control Statements – If , else, nested if, if-else ladders, Switch, while, do-while, for, for-each, break, continue.

1 Explain public static void main. In detail. **3/4**

OR

(ii) Method main is a public static method. Justify. **(Dec-15) [LJIET] .**

Ans:

- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is **no need to create object** to **invoke** the **static method**. The main method is executed by the JVM, **so it doesn't require to create object to invoke the main method**.
- **void** is the **return type** of the method, it means it **doesn't return** any value.
- **main** represents **startup function** of the program.
- **String[] args** is used for **command line argument.** String is the class and args is array of String object

2 (i) Explain short circuited operators. **(Dec-15) [LJIET] .** **3/4**

The `&&` and `||` operators perform *Conditional-AND* and *Conditional-OR* operations on two boolean expressions. **These operators exhibit "short-circuiting" behavior, which means that the second operand is evaluated only if needed.**

`&&` Conditional-AND - **Short circuit And**

`||` Conditional-OR - **Short circuit OR**

Operator	Description [A=true,B=false]
1	**&& (logical and)** Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. **Example** (A && B) is false.
2	**|| (logical or)** Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true. **Example** (A || B) is true.

Consider Ex. if(A || B) ----- ||(OR) operator results in true when A is true, no matter what B is.

- if(A &&B) ----- AND(&&) operator results in false when A is false, no matter what B is.

&& and || will not bother to evaluate the right-hand operand when the outcome of the expression can be determined by the left operand alone.

This is very useful when the right-hand operand depends on the left one being true or false in order to function properly.

See in this example:

Example:

```
class ShortCircuitDemo

        Public Static void main(String args[]){
          int a=5;
         int d=10;
         if( (d!=0) && ((a/d)>2))
         {}
    }
}
```

In example 2nd conditional expression - (a/d) never execute when d = 0. So in program there is no possibility of divide by zero error.

1 Explain short circuited operators and shift operators. **(June-11, May-13, June-14) [LJIET]** **3/4/7**

Ans : short circuited operators – above one (Topic-2 – Q-2)

shift operators : It is bit wise operator.

1	<< **(left shift)** Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand **Example:** A=4; A << 1 will give 8 (Shift bit value of 4 left one times) A << 2 will give 16 (Shift bit value of 4 left two times)
2	>> **(right shift – signed shift)** Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. >> is the arithmetic (or **signed**) right shift operator. **Example: :** A=4; A >> 1 will give 2 (Shift bit value of 4 right one times) A >> 2 will give 1(Shift bit value of 4 left right times) A=-4; A >> 1 will give -2 (Shift bit value of 4 right one time)
3	>>> **(zero fill right shift – unsigned shift)** Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros. >>> is the logical (or **unsigned**) right shift operator. **Example:** A= -4; A >>> 1 will give 2 (Shift bit value of 4 right one time but fill MSB bit with 0 – so it result into larger value 2147483646)

2 What is variable? How can we define variable in java? Also list rules for valid variable names. **(May-15) [LJIET]** **7**

A **variable** provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must **declare** all variables before they can be used. The basic form of a variable declaration is shown here:

```
data type variable [ = value][, variable [= value] ...] ;
```

Here *data type* is one of Java's datatypes and *variable* is the name of the variable. To declare more than one variable of the specified type, you can use a comma-separated list.

Following are valid examples of variable declaration and initialization in Java:

```
int a, b, c;          // Declares three ints, a, b, and c.
int a = 10, b = 10;   // Example of initialization
byte B = 22;          // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a';         // the char variable a iis initialized with value 'a'
```

Rules variable names: An variable name 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**. Variable name **can not be keyword**. Again, Java is case-sensitive, so VALUE is a different variable than Value.

valid variables are: AvgTemp count a4 $test this_is_ok classA

Invalid variable: 2count high-temp Not/ok class for

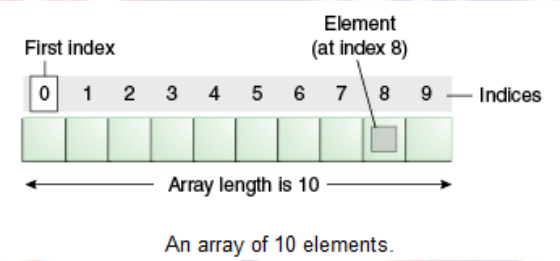
UNIT-2 ARRAY AND STRING:

TOPIC:1 (Array)

Single and Multidimensional Array

1 Explain array implementation in Java. **(Dec-15) [LJIET]** **3/4**

An *array* is a container object that holds a **fixed number** of values of **a single type**. The length of an array is established when the array is created. After creation, its length is **fixed**.



An array of 10 elements.

Syntax:

dataType[] arrayRefVar; // preferred way. **OR**

dataType arrayRefVar[]; // works but not preferred way.

Example:

// declares an array of integers

int[] anArray; OR int anArray[];

// allocates memory for 10 integers

anArray = new int[10];

// declares and allocation of memory together

int[] anArray = new int[10];

// Array declaration and initialization

int[] anArray = { 100,200,300,400, 500, 600,700, 800, 900,1000};

Property of Array: length

int[] anArray = { 100,200,300,400, 500, 600,700, 800, 900,1000};

anArray.length --→ use to dispay size of the array

Iteration Of Array: Using Example [Find Average of 4 integer nos]

```
public class TestArray {

   public static void main(String[] args) {
      int[] myList = {1, 2, 3, 3};

      // Print all the array elements
      for (int i = 0; i < myList.length; i++) {
         System.out.println(myList[i] + " ");
      }

      // Summing all elements
      double total = 0;
      for (int i = 0; i < myList.length; i++) {
         total += myList[i];
      }
      System.out.println("Total is " + total);
      double avg = total/myList.length;
      System.out.println("AVG is " + avg);
   }
}
```

Output:

```
1
2
3
3
Total is 9.0
```

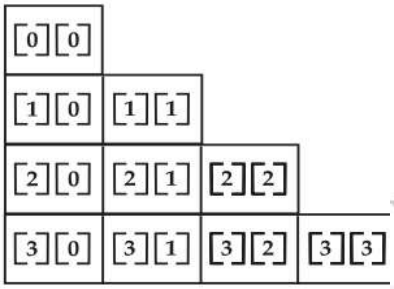
AVG is 2.25

2 Explain Ragged Array With example **(Nov-11) [LJIET]** **3/4**

Ans*:Ragged arrays :*—an array with rows of **nonuniform length** is known as a *ragged array*. There is no requirement that all rows in a two-dimensional array have the same length.

You can use Java arrays to create ragged arrays without a problem.

```
int[][] raggedArray = new int[n][];
for (int i=0; i<raggedArray.length; i++) {
   raggedArray[i] = new int[i+1];
}
```



or, irregular) multidimensional arrays i

1	Write a program that creates and initializes a four integer element array. Calculate and display the average of its values. **(Dec-15) [LJIET] .** **Ans: Unit-2 topic-1- Q-1**	**7**
	TOPIC:2 (String And Wrapper Classes) String class, StringBuffer class,Operations on string,, Command line argument, Use of Wrapper Class.	
1	Compare ii) String class and StringBuffer class. **(May-16) [LJIET] .**	**2/3**

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

1	Differentiate String class and StringBuffer class with explanation of its methods. . **(June-12,Jan-13, June-14,Dec-14) OR** Compare String with StringBuffer class. **(May-15) [LJIET] [For 7 marks write 7-10 methods only]**	**3/4/7**

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	static String format(String format, Object... args)	returns formatted string
4	static String format(Locale l, String format, Object... args)	returns formatted string with given locale
5	String substring(int beginIndex)	returns substring for given begin index
6	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index

7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value
8	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	returns a joined string
10	boolean equals(Object another)	checks the equality of string with object
11	boolean isEmpty()	checks if string is empty
12	String concat(String str)	concatinates specified string
13	String replace(char old, char new)	replaces all occurrences of specified char value
14	String replace(CharSequence old, CharSequence new)	replaces all occurrences of specified CharSequence
15	String trim()	returns trimmed string omitting leading and trailing spaces
16	String[] split(String regex)	returns splitted string matching regex
17	String[] split(String regex, int limit)	returns splitted string matching regex and limit
18	String intern()	returns interned string
19	int indexOf(int ch)	returns specified char value index
20	int indexOf(int ch, int fromIndex)	returns specified char value index starting with given index
21	int indexOf(String substring)	returns specified substring index
22	int indexOf(String substring, int fromIndex)	returns specified substring index starting with given index
23	String toLowerCase()	returns string in lowercase.
24	String toLowerCase(Locale l)	returns string in lowercase using specified locale.
25	String toUpperCase()	returns string in uppercase.
26	String toUpperCase(Locale l)	returns string in uppercase using specified locale.

Important methods of StringBuffer class

1. **public synchronized StringBuffer append(String s):** is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
2. **public synchronized StringBuffer insert(int offset, String s):** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
3. **public synchronized StringBuffer replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.

	4. **public synchronized StringBuffer delete(int startIndex, int endIndex):** is used to delete the string from specified startIndex and endIndex. 5. **public synchronized StringBuffer reverse():** is used to reverse the string. 6. **public int capacity():** is used to return the current capacity. 7. **public void ensureCapacity(int minimumCapacity):** is used to ensure the capacity at least equal to the given minimum. 8. **public char charAt(int index):** is used to return the character at the specified position. 9. **public int length():** is used to return the length of the string i.e. total number of characters. 10. **public String substring(int beginIndex):** is used to return the substring from the specified beginIndex. 11. **public String substring(int beginIndex, int endIndex):** is used to return the substring from the specified beginIndex and endIndex.	

2	Compare String with StringBuffer. Also write a program to count occurrence of character in a string. **(Dec-13) [LJIET]**
	Compare String with StringBuffer : Ans – 1
	Write a program to count occurrence of character in a string.
	class CharecterOccuranceCount {
	public static void main(String[] args) {
	String s = "javaisplatformindependent";
	int l = s.length();
	int count = 0;
	for (char ch = 'a'; ch <= 'z'; ch++) {
	count = 0;
	for (int i = 0; i < l; i++) {
	char c = s.charAt(i);
	if (c == ch) {
	count++;
	}
	}
	System.out.println("The Occurrence of - "+ ch+" -is: " + count);
	}
	}
	}
	Output:
	The Occurrence of - a -is: 3
	The Occurrence of - b -is: 0
	The Occurrence of - c -is: 0
	The Occurrence of - d -is: 2
	The Occurrence of - e -is: 3
	The Occurrence of - f -is: 1
	The Occurrence of - g -is: 0
	The Occurrence of - h -is: 0
	The Occurrence of - i -is: 2
	The Occurrence of - j -is: 1
	The Occurrence of - k -is: 0
	The Occurrence of - l -is: 1
	The Occurrence of - m -is: 1
	The Occurrence of - n -is: 3
		3/4/7

	The Occurrence of - o -is: 1 The Occurrence of - p -is: 2 The Occurrence of - q -is: 0 The Occurrence of - r -is: 1 The Occurrence of - s -is: 1 The Occurrence of - t -is: 2 The Occurrence of - u -is: 0 The Occurrence of - v -is: 1 The Occurrence of - w -is: 0 The Occurrence of - x -is: 0 The Occurrence of - y -is: 0 The Occurrence of - z -is: 0	

3	State whether any error exists in the following code. If so, correct the error and give output. .
	(Nov-11) [LJIET]
	class Test {
	public static void main(String args[]) {
	A a = new A();
	a.print();
	}
	}
	class A {
	String s;
	A(String s) {
	this.s = s;
	}
	public void print() {
	System.out.println(s);
	}
	}
	Output: Ans:
	Error : A a = new A(); --- Can not create object of A without using parameter.
	
	Corrected code:
	class Test {
	public static void main(String args[]) {
	A a = new A("5-CE");
	a.print();
	}
	}
	class A {
	String s;
	A(String s) {
	this.s = s;
	}
	public void print() {
	System.out.println(s);
	}
	}
		3/4

OR

```
class Test {
   public static void main(String args[]) {
      A a = new A();
      a.print();
   }
}
class A {
   String s;
A(){
      s= "5-CE";
}
   A(String s) {
      this.s = s;
   }
   public void print() {
      System.out.println(s);
   }
}
```

4 Write a java program to do sum of command line argument passed two Double numbers. **(Dec-15) [LJIET]** **7**

```
public class SumCommandLineDoubleValue {
    public static void main(String[] args) {
        double sum = 0;
        for (int i = 0; i < args.length; i++) {

            double d = Double.parseDouble(args[i]);
            sum = sum+d;
        }
        System.out.println("Sum = " + sum);
    }
}
```

Run : **java SumCommandLineDoubleValue 1.0 2.3 3.7**

Output: Sum = 7.0

5 What is Wrapper class in Java? Explain with examples.**(May-16) [LJIET]** **7**

Wrapper class in java provides the **mechanism *to convert primitive into object and object into primitive*.**

Another Use: Parsing string into primitive and Objects.

Ex. : int i = Integer.parseInt("1");

"1" --→ in form of String , it is not int. Syntax use to convert string contain 1 into primitive data type int i = 1;

Since J2SE 5.0, **autoboxing** and **unboxing** feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known and autoboxing and vice-versa unboxing.

One of the eight classes of *java.lang* package are known as wrapper class in java. The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Note: You can write example of Tpoic-2 Ans -4 (sum of command line double value)

Example: Wrapper class Example: Wrapper to Primitive:

```
class WrapperExample2{
        public static void main(String args[]){
                //Converting Integer to int
                Integer a=new Integer(5);
                int i=a.intValue();//converting Integer to int
                int j=a;//unboxing, now compiler will write a.intValue() internally

                System.out.println(a+" "+i+" "+j);
        }
}
```

Output : 5 5 5