CBCS SCHEME

C.M.R. INSTITUTE OF TECHNOLOGY CMR BANGALORE

USN | | | | | | | | | |

BCS613D

Sixth Semester B.E./B.Tech. Degree Examination, June/July 2025
Advanced Java

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	What is a collection framework? Explain the methods defined by collection interface.	10	L2	CO1
	b.	Develop a java program to create an ArrayList of objects of type string. Add any five strings, display size and contents of list. Remove any two strings and display size and contents.	10	L3	CO1
		OR			
Q.2	a.	Explain the constructors of tree set class and develop a java program to create Treeset collection and access it via an iterator.	10	L3	CO1
	b.	Explain any four legacy classes of Java's collection framework.	10	L2	CO1
		Module – 2			
Q.3	a.	What is string in Java? Illustrate java program that demonstrates any four constructors of string class.	10	L2	CO2
	b.	Compare between equals( ) and = = with respect to string comparison.	4	L2	CO2
	c.	Explain the following character extraction methods : i) charAt( ) ii) getChars( ) iii) toCharArray( )	6	L2	CO2
		OR			
Q.4	a.	Explain any four string modification methods of string class.	10	L2	CO2
	b.	Explain the following methods of string buffer class: i) append( ) ii) insert( ) iii) reverse( ) iv) replace( )	10	L2	CO2
		Module – 3			
Q.5	a.	Explain the key features of swing and also develop the program.	10	L3	CO3
	b.	Build a program to demonstrate an icon-based button. Each button displays an icon that represents the flag of a country. When a button is pressed, the name of that country is displayed in the label.	10	L3	CO3
		OR			
Q.6	a.	Explain the event handling mechanism used by swing and also develop the program.	10	L3	CO3
	b.	Illustrate the use of radio buttons and also develop the program.	10	L3	CO3

		Module – 4	M	L	C
Q.7	a.	List and explain the core classes and interfaces in javax.servlet package.	10	L2	CO4
	b.	Develop a java servlet program to accept two parameters from webpage, find the sum of them and display the result in the webpage. Also give necessary html script to create web page.	10	L3	CO4
		OR			
Q.8	a.	Define JSP. Explain different JSP tags with suitable example program.	8	L2	CO4
	b.	Explain the life cycle of a servlet.	4	L2	CO4
	c.	What is a cookie? Listout the methods defined by cookie. Develop a Java program to add a cookie.	8	L3	CO4
		Module – 5			
Q.9	a.	Explain four types of JDBC drivers.	5	L2	CO5
	b.	Construct a code snippet to describe the various steps involved in JDBC process.	10	L3	CO5
	c.	Explain database meta data and result set metadata.	5	L2	CO5
		OR			
Q.10	a.	What is statement object in JDBC? Explain the following statement objects: i) Callable statement object ii) Prepared statement object.	10	L2	CO5
	b.	Develop a java program to execute a database transaction.	6	L3	CO5
	c.	Show any two syntax of establishing a connection to database.	4	L2	CO5

CMRIT LIBRARY BANGALORE - 560 037

* * * * *

VTU-07-07-2025 01:32:36pm CR - CR - CR

Solution to June-July 2025 Advanced Java BCS613D Question Paper

Q1.

a. What is the Collection framework? Explain the methods defined by the collection interface. 10 Marks

Ans:

Collection Framework:

- The Collections Framework is a sophisticated hierarchy of interfaces and classes that provide state-of-the-art technology for managing groups of objects.
- It is a unified architecture for representing and manipulating collections.
- A collection is an object that groups multiple elements into a single unit.
- The framework provides reusable data structures and algorithms to perform operations like adding, removing, sorting, and searching elements.

The Collection Interface:

The Collection interface is the foundation upon which the Collections Framework is built because it must be implemented by any class that defines a collection. Collection is a generic interface that has this declaration:

interface Collection<E>

Here, E specifies the type of objects that the collection will hold.

Methods of Collection Interface:

1. **boolean add (E obj) and**
2. **boolean addAll (Collection<? Extends E> c):**
   add() method adds Objects to a collection. It takes an argument of type E, which means that objects added to a collection must be compatible with the type of data expected by the collection.
   You can add the entire contents of one collection to another by calling addAll( ).
3. **boolean remove (Object obj)**
4. **boolean removeAll (Collection<?> c)**
   remove() method removes an object from the collection.
   To remove a group of objects, call removeAll( ).
5. **Boolean retainAll (Collection <?> c)**
   You can remove all elements except those of a specified group by calling retainAll( ).
6. **Default boolean removeIf (Predicate<? Super E> predicate)**
   To remove an element only if it satisfies some condition, you can use removeIf( ).
7. **void clear ()**
   To empty a collection, call clear( ).
8. **Boolean contains (Object obj)**
9. **Boolean containsAll (Collection <?> c)**
   You can determine whether a collection contains a specific object by calling contains( ). To determine whether one collection contains all the members of another, call containsAll( ).
10. **boolean isEmpty()**

You can determine when a collection is empty by calling isEmpty( ).

11. **Int size()**
    The number of current elements in a collection can be determined by calling size( ).
12. **Default <T> T[ ] toArray (IntFunction <T[ ]> arrayGen)**
13. **Object[ ] toArray()**
    The toArray( ) methods return an array that contains the elements stored in the invoking collection. The first returns an array of elements that have the same type as the array specified as a parameter. The second returns an array of Objects. Normally, the first form is more convenient because it returns the desired array type.
    Often, processing the contents of a collection by using array-like syntax is advantageous. By providing a pathway between collections and arrays, you can have the best of both worlds.
14. **boolean equals (Object obj)**
    Two collections can be compared for equality by calling equals( ). The precise meaning of "equality" may differ from collection to collection. For example, you can implement equals( ) so that it compares the values of elements stored in the collection. Alternatively, equals( ) can compare references to those elements.
15. **Iterator<E> iterator()**
16. **Default Spliterator<E> spliterator()**
    iterator( ) returns an iterator to a collection.
    The spliterator( ) method returns a spliterator to the collection. Iterators are frequently used when working with collections.
17. **stream( )**
18. **parallelStream( )**
    The stream() and parallelStream() methods return a Stream that uses the collection as a source of elements.

b. Develop a java program to create an ArrayList of objects for type String. Add any five Strings display size and contents of the list. Remove any two strings and display size and contents. 10 Marks

```
import java.util.ArrayList;

public class ArrayListExample {
    public static void main(String[] args) {
        // Create an ArrayList of Strings
        ArrayList<String> stringList = new ArrayList<>();

        // Add five strings
        stringList.add("Java");
        stringList.add("Python");
        stringList.add("C++");
        stringList.add("JavaScript");
        stringList.add("Ruby");

        // Display the size and contents
        System.out.println("Initial size: " + stringList.size());
        System.out.println("Initial contents: " + stringList);

        // Remove two strings
        stringList.remove("Python");
        stringList.remove("Ruby");

        // Display the updated size and contents
        System.out.println("Updated size: " + stringList.size());
        System.out.println("Updated contents: " + stringList);
    }
}
```

Output

```
Initial size: 5
Initial contents: [Java, Python, C++, JavaScript, Ruby]
Updated size: 3
Updated contents: [Java, C++, JavaScript]
```

Q2.

a. Explain the constructors of TreeSet class and develop a Java program to create TreeSet collection and access it via an iterator. 10 Marks

Ans:

Constructors of TreeSet Class:

TreeSet has the following constructors:

1. TreeSet( )
2. TreeSet(Collection<? extends E> c)
3. TreeSet(Comparator<? super E> comp)
4. TreeSet(SortedSet<E> ss)

1. **TreeSet( )**

The first form constructs an empty tree set that will be sorted in ascending order according to the natural order of its elements.

2. **TreeSet(Collection<? extends E> c)**

The second form builds a tree set that contains the elements of c.

3. **TreeSet(Comparator<? super E> comp)**

The third form constructs an empty tree set that will be sorted according to the comparator specified by comp.

4. **TreeSet(SortedSet<E> ss)**

The fourth form builds a tree set that contains the elements of ss.

Java Program using TreeSet and Iterator

```
import java.util.Iterator;
import java.util.TreeSet;
```

```
public class TreeSetExample {
   public static void main(String[] args) {
      // Create a TreeSet of strings
      TreeSet<String> set = new TreeSet<>();

      // Add elements to the TreeSet
      set.add("Apple");
      set.add("Mango");
      set.add("Banana");
      set.add("Grapes");
      set.add("Orange");

      // Display the elements using Iterator
      System.out.println("Elements in TreeSet (sorted):");
      Iterator<String> iterator = set.iterator();
      while (iterator.hasNext()) {
         System.out.println(iterator.next());
      }
   }
}
```

Output

```
Elements in TreeSet (sorted):
Apple
Banana
Grapes
Mango
Orange
```

Explanation:

- TreeSet automatically sorts elements in natural order (alphabetical for strings).
- Iterator is used to access elements one by one.

b. Explain any four legacy classes of Java's Collections Framework. 10 Marks

Ans:

Legacy Classes in Java Collections Framework

Legacy classes are part of Java's original java.util package before the Collections Framework was introduced in **Java 2 (JDK 1.2)**. These classes were later retrofitted to implement collection interfaces.

Here are **four important legacy classes**:

---

1. Vector

- **Description**: A growable array of objects, similar to ArrayList, but **synchronized**, which means it's thread-safe.
- **Key Methods**: add(), remove(), elementAt(), size(), capacity().

Usage:

```
 Vector<String> v = new Vector<>();
v.add("Java");
v.add("Python");
System.out.println(v);
```

---

2. Stack

- **Description**: A subclass of Vector that represents a **Last-In-First-Out (LIFO)** stack.
- **Key Methods**: push(), pop(), peek(), empty(), search().

Usage:

```
 Stack<Integer> stack = new Stack<>();
stack.push(10);
stack.push(20);
System.out.println(stack.pop()); // Outputs 20
```

---

3. Hashtable

- **Description**: A key-value pair data structure like HashMap, but **synchronized**.
- **Key Methods**: put(), get(), containsKey(), containsValue(), remove().

Usage:

```
 Hashtable<Integer, String> ht = new Hashtable<>();
ht.put(1, "Java");
ht.put(2, "Python");
System.out.println(ht.get(1)); // Java
```

---

4. Enumeration

- **Description**: An interface used to iterate elements of legacy collections like Vector and Hashtable.
- **Key Methods**: hasMoreElements(), nextElement().

Usage:

```
 Vector<String> languages = new Vector<>();
languages.add("Java");
languages.add("C++");
```

```
Enumeration<String> e = languages.elements();
while (e.hasMoreElements()) {
   System.out.println(e.nextElement());
}
```

Q3.

a. What is String in Java? Illustrate Java program that demonstrates any four constructors of String class. 10 Marks

Ans:

In Java, a **String** is a class from the java.lang package used to represent a **sequence of characters**. It is **not a primitive data type**, but a **reference type** (i.e., an object).

The String Constructors

String support following constructors:

1. Default Constructor
2. Constructor with an array of characters as a parameter
3. Constructor with a subrange of character array as parameter
4. Constructor with String object as parameter
5. Constructor with byte array as parameter:
6. Constructor with range of byte array as parameter
7. Constructor with StringBuffer as a parameter
8. Constructor with StringBuilder as a parameter
9. Constructor to support extended Unicode character set

1. Default Constructor:

- To create an empty String, call the default constructor. For example,
  String s = new String();
  will create an instance of String with no characters in it.

2. Constructor with an array of characters as a parameter

- To create a String initialized by an array of characters, we use the constructor shown here:
  String(char chars[ ])
  Here is an example:
  char chars[] = { 'a', 'b', 'c' };
  String s = new String(chars);
  This constructor initializes s with the string "abc".

3. Constructor with a subrange of character array as parameter:

- You can specify a subrange of a character array as an initializer using the following constructor:
  String(char chars[ ], int startIndex, int numChars)
- startIndex specifies the index at which the subrange begins, and numChars specifies the number of characters to use.
- Here is an example:
  char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };

```
String s = new String(chars, 2, 3);
```
This initializes s with the characters - cde.

4. Constructor with String object as parameter:

- You can construct a String object that contains the same character sequence as another String object using this constructor:
  String(String strObj)
- Here, strObj is a String object. Consider this example:

```
// Construct one String from another.
class MakeString {
        public static void main(String args[]) {
                char c[] = {'J', 'a', 'v', 'a'};
                String s1 = new String(c);
                String s2 = new String(s1);
                System.out.println(s1);
                System.out.println(s2);
        }
}
```

- The output from this program is as follows:

```
Java
Java
```
s1 and s2 contain the same string.

Example program to illustrate all the constructors of class String:

```
public class StringConstructorsDemo {
   public static void main(String[] args) {
      // 1. Default Constructor
      String emptyString = new String();
      System.out.println("1. Default Constructor: '" + emptyString + "'");

      // 2. Constructor with an array of characters as a parameter
      char[] charArray = {'H', 'e', 'l', 'l', 'o'};
      String strFromCharArray = new String(charArray);
      System.out.println("2. Constructor with char array: " + strFromCharArray);

      // 3. Constructor with a subrange of character array as parameter
         String strFromSubCharArray = new String(charArray, 1, 3); // Starts at index 1,
length 3
         System.out.println("3. Constructor with subrange of char array: " +
strFromSubCharArray);

      // 4. Constructor with String object as parameter
      String originalString = "Java Programming";
      String strFromString = new String(originalString);
      System.out.println("4. Constructor with String object: " + strFromString);
      // 5. Constructor with byte array as parameter
      byte[] byteArray = {74, 97, 118, 97}; // ASCII values for 'J', 'a', 'v', 'a'
      String strFromByteArray = new String(byteArray);
      System.out.println("5. Constructor with byte array: " + strFromByteArray);

      // 6. Constructor with range of byte array as parameter
         String strFromSubByteArray=new String(byteArray, 1, 2);// Starts at index 1,
length 2
      System.out.println("6.Constructor with subrange of byte array:"+
strFromSubByteArray);

      // 7. Constructor with StringBuffer as a parameter
      StringBuffer stringBuffer = new StringBuffer("StringBuffer Example");
      String strFromStringBuffer = new String(stringBuffer);
      System.out.println("7. Constructor with StringBuffer: " + strFromStringBuffer);

      // 8. Constructor with StringBuilder as a parameter
      StringBuilder stringBuilder = new StringBuilder("StringBuilder Example");
      String strFromStringBuilder = new String(stringBuilder);
      System.out.println("8. Constructor with StringBuilder: " + strFromStringBuilder);

      // 9. Constructor to support extended Unicode character set
      int[] codePoints = {0x1F600, 0x1F604, 0x1F609}; // Unicode for 😀, 😄, 😉
      String strFromUnicode = new String(codePoints, 0, codePoints.length);
      System.out.println("9. Constructor with extended Unicode: " + strFromUnicode);
   }
}
```

b. Compare between equals() and == with respect to String comparison 4 Marks

Ans:

equals( ) Versus ==

- The equals( ) method compares the characters inside a String object.
- The == operator compares two object references to see whether they refer to the same instance.
- The following program shows how two different String objects can contain the same characters, but references to these objects will not compare as equal:

```
// equals() vs ==
class EqualsNotEqualTo {
        public static void main(String args[]) {
                String s1 = "Hello";
                String s2 = new String(s1);
                System.out.println(s1 + " equals " + s2 + " -> " +
                s1.equals(s2));
                System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2));
        }
}
```

- The variable s1 refers to the String instance created by "Hello".
- The object referred to by s2 is created with s1 as an initializer.
- Thus, the contents of the two String objects are identical, but they are distinct objects.
- This means that s1 and s2 do not refer to the same objects and are, therefore, not ==, as is shown here by the output of the preceding example:

  ```
  Hello equals Hello -> true
  Hello == Hello -> false
  ```

Example:
String s1 = "Hello";
String s2 = "Hello";
s1.equals(s2) - true
S1 == s2 - true

Example:
String a = "a";
String b = new String("a");
System.out.println(a == b); // <-- false
System.out.println(a.equals(b)); // <-- true

c. Explain the following character extraction methods 6 Marks
   i. charAt()
   ii. getChars()
   iii. toCharArray()

Ans:

1. charAt( )

- To extract a single character from a String, you can refer directly to an individual character via the charAt( ) method. It has this general form:

  char charAt(int where)
- where is the index of the character that you want to obtain.
- The value of where must be nonnegative and specify a location within the string.
- charAt( ) returns the character at the specified location.
- For example,

  ```
  char ch;
  ch = "abc".charAt(1);
  ```

  assigns the value b to ch.
- Programs:
  1. Count Vowels in a String
  2. Find the First Non-Repeating Character
  3. Extract Digits from a String
  4. Check for Balanced Parentheses

2. getChars( )

- If you need to extract more than one character at a time, you can use the getChars( ) method. It has this general form:

  void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)
- Here, sourceStart specifies the index of the beginning of the substring, and sourceEnd specifies an index that is one past the end of the desired substring.

- Thus, the substring contains the characters from sourceStart through sourceEnd–1.
- The array that will receive the characters is specified by target.
- The index within the target at which the substring will be copied is passed in targetStart.
- **Care must be taken to ensure that the target array is large enough to hold the number of characters in the specified substring.**

```
// Program to demonstrates getChars( ):
class getCharsDemo {
      public static void main(String args[]) {
                                    String s = "This is a demo of the getChars method.";
                                    int start = 10;
                                    int end = 14;
                                    char buf[] = new char[end - start];
                                    s.getChars(start, end, buf, 0);
                                    System.out.println(buf);
      }
}
```

Output:

```
demo
```

1. toCharArray( )

- toCharArray( ) converts all the characters in a String object into a character array.
- It returns an array of characters for the entire string. It has this general form:

  char[ ] toCharArray( )

- This function is provided as a convenience since it is possible to use getChars( ) to achieve the same result.

```
String s = "This is a demo of the getChars method.";
char target[] = new char[s.length()];
s.getChars(0, s.length(), target, 0)
Same as,  target = s.toCharArray();
```

- **Example program:**

```
public class ToCharArrayExample {
   public static void main(String[] args) {
      // Define a string
      String str = "Hello, Java!";
      // Convert string to a character array
      char[] charArray = str.toCharArray();
      // Print each character using a loop
      System.out.println("Characters in the array:");
      for (char ch : charArray) {
         System.out.print(ch + " ");
      }
   }
}
```

Q4.

a. Explain any four String modification methods of String class 10 Marks

String objects are immutable, whenever you want to modify a String, you must either copy it into a **StringBuffer or StringBuilder** or use a String method that constructs a new copy of the string with your modifications.

Following are String modification methods:

1. substring()
2. concat()
3. replace()
4. trim() and strip()

1. substring( )

- You can extract a substring using **substring( ).** It has two forms
- The first is

  String substring(int startIndex)

  Here, startIndex specifies the index at which the substring will begin. This form returns a copy of the substring that starts at startIndex and runs to the end of the invoking string.

- The second form of substring( ) allows you to specify both the beginning and ending index of the substring:

  String substring(int startIndex, int endIndex)

  StartIndex specifies the beginning index, and endIndex specifies the stopping point.
- **The string returned contains all the characters from the beginning index, up to, but not including, the ending index.**

Program to demonstrate substring( ) to replace all instances of one substring with another within a string:

```
// Substring replacement using substring().
class StringReplace {
        public static void main(String args[]) {
                String org = "This is a test. This is, too.";
                String search = "is";
                String sub = "was";
                String result = "";
                int i;
                do { // replace all matching substrings
                        System.out.println(org);
                        i = org.indexOf(search);
                        if(i != -1) {
                                result = org.substring(0, i);
                                result = result + sub;
                                result = result + org.substring(i + search.length());
                                org = result;
                        }
                } while(i != -1);
        }
}
```

The output from this program is shown here:

```
This is a test. This is, too.
Thwas is a test. This is, too.
Thwas was a test. This is, too.
Thwas was a test. Thwas is, too.
Thwas was a test. Thwas was, too.
```

2. concat( )

- You can concatenate two strings using concat( ), shown here:

  String concat(String str)
- This method creates a new object that contains the invoking string with the contents of str appended to the end.
- **concat( ) performs the same function as +.**
- For example,

  ```
  String s1 = "one";
  String s2 = s1.concat("two");
  ```

  puts the string "onetwo" into s2.
- It generates the same result as the following sequence:

  ```
  String s1 = "one";
  String s2 = s1 + "two";
  ```

3. replace( )

- The replace( ) method has two forms.
- The first replaces all occurrences of one character in the invoking string with another character. It has the following general form:

  String replace(char original, char replacement)
- The original specifies the character to be replaced by the character specified by the replacement.
- The resulting string is returned. For example,

  ```
  String s = "Hello".replace('l', 'w');
  ```

  puts the string "**Hewwo**" into string s.
- The second form of replace( ) replaces one character sequence with another. It has this general form:

  String replace(CharSequence original, CharSequence replacement)

Program to demonstrate replace() to replace one character sequence with another.

```
public class ReplaceExample2 {
   public static void main(String args[]){
      String s1="This is a test. This is, too.";
      String replaceString=s1.replace(" is"," was");
//replaces all occurrences of "is" to "was"
      System.out.println(replaceString);
   }
}
```

Output:

```
This was a test. This was, too.
```

4. trim( ) and strip()

- The trim( ) method returns a copy of the invoking string from which any leading and trailing whitespace has been removed.
- It has this general form:

  String trim( )
- Here is an example:

  String s = "   Hello World   ".trim();

  This puts the string "Hello World" into s.
- The trim( ) method is quite useful when you process **user commands.**
- For example, the following program prompts the user for the name of a state and then displays that state's capital.
- It uses trim( ) to remove any leading or trailing whitespace that the user may have inadvertently entered.

Program to demonstrate trim()

```
// Using trim() to process commands.
import java.io.*;
class UseTrim {
        public static void main(String args[]) throws IOException
        {
                //Create a BufferedReader using System.in
                BufferedReader br = new BufferedReader
                (new InputStreamReader(System.in, System.console().charset()));
                String str;
                System.out.println("Enter 'stop' to quit.");
                System.out.println("Enter State: ");
                do {
                        str = br.readLine();
                        str = str.trim(); // remove whitespace
                        if(str.equals("Illinois"))
                        System.out.println("Capital is Springfield.");
                        else if(str.equals("Missouri"))
                        System.out.println("Capital is Jefferson City.");
                        else if(str.equals("California"))
                        System.out.println("Capital is Sacramento.");
                        else if(str.equals("Washington"))
                        System.out.println("Capital is Olympia.");
                        // …
                } while(!str.equals("stop"));
        }
}
```

strip(), stripLeading(), stripTrailing():

- Beginning with JDK 11, Java provides the methods **strip(), stripLeading() and stripTrailing().**

strip():

- strip() method removes **all whitespace characters** (as defined by Java) from the beginning and end of the invoking string and returns the result.
- These whitespace characters include **spaces, tabs, carriage returns and line feeds**.

stripLeading() and stripTrailing():

- These two methods delete whitespace characters from the start or end of the invoking string and return the result.

b. Explain the following methods of StringBuffer class 10 Marks
   i. append()
   ii. insert()
   iii. reverse()
   iv. replace()

Ans:

i. append( )

- The append( ) method concatenates the string representation of any other type of data to the end of the invoking StringBuffer object.
- It has several overloaded versions. Here are a few of its forms:

  StringBuffer append(String str)
  StringBuffer append(int num)
  StringBuffer append(Object obj)

- **The string representation of each parameter is obtained. Then,** the result is appended to the current StringBuffer object.
- The buffer itself is returned by each version of append( ).
- This allows subsequent calls to be chained together, as shown in the following example:

```
// Demonstrate append().
class appendDemo {
	public static void main(String args[]) {
		String s;
		int a = 42;
		StringBuffer sb = new StringBuffer(40);
		s = sb.append("a = ").append(a).append("!").toString();
		System.out.println(s);
	}
}
```

The output of this example is shown here:

```
a = 42!
```

ii. insert( )

- The insert( ) method inserts one string into another.
- It is overloaded to accept values of all the primitive types, plus Strings, Objects, and CharSequences.
- Like append( ), it obtains the string representation of the value it is called with.
- This string is then inserted into the invoking StringBuffer object.

- These are a few of its forms:
  StringBuffer insert(int index, String str)
  StringBuffer insert(int index, char ch)
  StringBuffer insert(int index, Object obj)
- Here, index specifies the index at which point the string will be inserted into the invoking StringBuffer object.

The following sample program inserts "like" between "I" and "Java":

```
// Demonstrate insert().
        class insertDemo {
                public static void main(String args[]) {
                StringBuffer sb = new StringBuffer("I Java!");
                sb.insert(2, "like ");
                System.out.println(sb);
        }
}
```

Output:

```
I like Java!
```

iii. reverse( )

- You can reverse the characters within a StringBuffer object using reverse( ), shown here:
  StringBuffer reverse( )
- This method returns the reverse of the object on which it was called.
- The following program demonstrates reverse( ):

```
// Using reverse() to reverse a StringBuffer.
class ReverseDemo {
        public static void main(String args[]) {
                StringBuffer s = new StringBuffer("abcdef");
                System.out.println(s);
                s.reverse();
                System.out.println(s);
        }
}
```

Output:

```
abcdef
fedcba
```

iv. replace( )

- You can replace one set of characters with another set inside a StringBuffer object by calling replace( ).
- Its signature is shown here:
  StringBuffer replace(int startIndex, int endIndex, String str)
- The substring being replaced is specified by the indexes startIndex and endIndex. Thus, the substring at startIndex through endIndex–1 is replaced.
- The replacement string is passed in str. The resulting StringBuffer object is returned.

The following program demonstrates replace( ):

```
// Demonstrate replace()
class replaceDemo {
        public static void main(String args[]) {
                StringBuffer sb = new StringBuffer("This is a test.");
                sb.replace(5, 7, "was");
                System.out.println("After replace: " + sb);
        }
}
```
Here is the output:
After replace: This was a test.

Q5.

a. Explain the key features of Swing and also develop the program. 10 Marks

Two Key Swing Features

Swing's power comes from two defining features that set it apart from its predecessor, AWT, and make it a cornerstone of Java GUI development. These are:

1. Lightweight Components
2. Pluggable Look and Feel (PLAF)

1. Lightweight Components

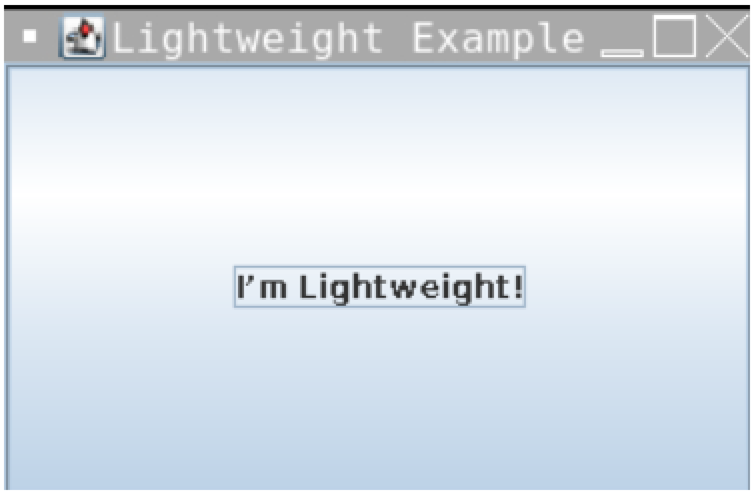
- Swing components (e.g., JButton, JLabel, JTextField) are called "lightweight" because they don't rely on the operating system's native GUI system to draw or manage them. Instead, they're entirely implemented in Java and rendered using Java's own graphics capabilities.
- Contrast this with AWT's "heavyweight" components (e.g., Button, TextField), which depend on the OS (like Windows or macOS) to provide their look and behavior.

Working:
- Lightweight components are painted directly onto the screen by Java's Graphics class, typically inside an AWT top-level container like JFrame or JDialog.
- They bypass the OS's native rendering, giving Swing full control over their appearance and behavior.
- Example: A JButton doesn't call a Windows button API—it's drawn pixel-by-pixel by Java.

Importance:
- **Portability:** Because Swing doesn't use native OS components, a JButton looks and works the same on Windows, macOS, Linux, or any Java-supported platform. This fulfills Java's "write once, run anywhere" promise for GUIs, unlike AWT's inconsistent look.
- **Flexibility:** Since Java controls the drawing, Swing can customize components (e.g., shapes, colors) beyond what the OS offers.
- **Reduced Overhead:** Lightweight components avoid the performance cost of coordinating with the OS for every button or label.

Example

```
import javax.swing.*;

public class LightweightDemo {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Lightweight Example");
      frame.setSize(300, 200);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JButton button = new JButton("I'm Lightweight!");
      frame.add(button); // JButton drawn by Java, not the OS

      frame.setVisible(true);
   }
}
```



- Explanation: The JButton here is a lightweight component. Its appearance (e.g., rounded edges, shading) is handled by Swing's rendering engine, not the OS. On any system, it'll look like Swing's default style (usually the "Metal" look).

Trade-offs
- Early Swing versions were slower than AWT due to Java handling all rendering, but modern Java optimizations have minimized this gap.
- Mixing lightweight Swing components with heavyweight AWT ones can cause display issues (e.g., overlapping quirks), so it's best to stick to one or the other.

2. Pluggable Look and Feel (PLAF)

- PLAF allows you to change the visual style (or "look and feel") of a Swing application without altering its code. Swing can mimic the OS's native appearance (e.g., Windows, macOS) or use custom styles (e.g., Metal, Nimbus).
- This is a huge leap from AWT, where the look was fixed to whatever the OS provided.

Working:
- Swing separates a component's functionality (e.g., a button's click action) from its appearance (e.g., how it's drawn). This separation is rooted in the Model-View-Controller (MVC) design internally.
- The "look and feel" is managed by a set of classes in the javax.swing.plaf package. You can switch these classes at runtime using UIManager.setLookAndFeel().

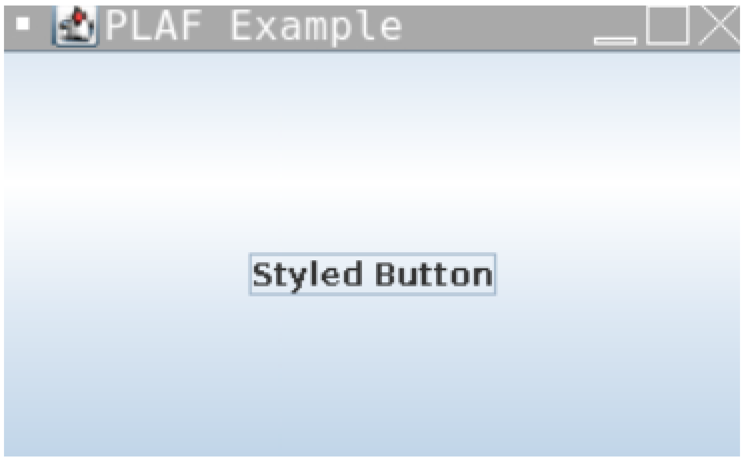
- Available options include:
  - Metal: Swing's default cross-platform look.
  - Windows: Mimics Windows-style GUIs.
  - Motif: An older UNIX-like style.
  - Nimbus: A modern, polished look (added in later Java versions).
  - Custom looks: Developers can create their own.

Importance:
- User Experience: You can match the app's style to the user's OS (e.g., Windows look on Windows) for familiarity or keep it consistent across platforms.
- Customization: PLAF lets you brand an app with a unique style, unlike AWT's rigid native dependency.
- No Code Changes: Switching looks doesn't require rewriting the app—just a single line of code.

Example

```
import javax.swing.*;
import java.awt.*;

public class LookAndFeelDemo {
   public static void main(String[] args) {
      try {
         // Set Windows look and feel (works on Windows OS)

UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
);
      } catch (Exception e) {
         e.printStackTrace();
      }

      JFrame frame = new JFrame("PLAF Example");
      frame.setSize(300, 200);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JButton button = new JButton("Styled Button");
      frame.add(button);
```

```
                frame.setVisible(true);
            }
        }
```



- Explanation:
  - This code sets the Windows look and feel (if run on Windows). The JButton will look like a native Windows button, but it's still lightweight—Swing just mimics the style.
  - Change the UIManager.setLookAndFeel() argument to "javax.swing.plaf.metal.MetalLookAndFeel" for the Metal look, and the button's style will switch without altering the rest of the code.
- Output: A window with a button styled according to the chosen look (Windows or Metal, depending on the setting).

Listing Available Looks
You can see all installed look-and-feel options like this:

```
import javax.swing.*;

public class ListLookAndFeel {
    public static void main(String[] args) {
        UIManager.LookAndFeelInfo[] looks = UIManager.getInstalledLookAndFeels();
        for (UIManager.LookAndFeelInfo look : looks) {
            System.out.println(look.getName() + ": " + look.getClassName());
        }
    }
}
```

- Speculative Output (depends on Java version):

```
  Metal: javax.swing.plaf.metal.MetalLookAndFeel
  Nimbus: javax.swing.plaf.nimbus.NimbusLookAndFeel
  Windows: com.sun.java.swing.plaf.windows.WindowsLookAndFeel
  ...
```

Trade-offs
- Some looks (e.g., Windows) only work on their respective OS.
- Switching look and feel at runtime can be tricky if the app's layout isn't designed flexibly.

Why These Two Features Matter
- Lightweight Components: Give Swing its portability and customization power, breaking free from AWT's OS dependency.
- Pluggable Look and Feel: Add versatility, letting developers tailor the app's appearance to users or branding needs.
- Together, they make Swing a robust, adaptable GUI toolkit, far surpassing AWT's capabilities.

b. Build a program to demonstrate an icon based button. Each button displays an icon that represents the flag of a country. When a button is clicked the name of that country is displayed in the label. 10 Marks

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class FlagButtonDemo extends JFrame implements ActionListener {
   JLabel countryLabel;
   JButton indiaBtn, usaBtn, ukBtn;

   public FlagButtonDemo() {
      setTitle("Flag Button Demo");
      setSize(400, 250);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setLayout(new FlowLayout());

      // Load images (ensure flag icons are available in the project directory)
      ImageIcon indiaIcon = new ImageIcon("india.png");
      ImageIcon usaIcon = new ImageIcon("usa.png");
      ImageIcon ukIcon = new ImageIcon("uk.png");

      // Buttons with icons
      indiaBtn = new JButton(indiaIcon);
      usaBtn = new JButton(usaIcon);
      ukBtn = new JButton(ukIcon);

      // Add action listeners
      indiaBtn.addActionListener(this);
      usaBtn.addActionListener(this);
      ukBtn.addActionListener(this);

      // Label
```

```
        countryLabel = new JLabel("Click a flag to see the country");

        // Add components to frame
        add(indiaBtn);
        add(usaBtn);
        add(ukBtn);
        add(countryLabel);

        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == indiaBtn) {
            countryLabel.setText("India");
        } else if (e.getSource() == usaBtn) {
            countryLabel.setText("United States of America");
        } else if (e.getSource() == ukBtn) {
            countryLabel.setText("United Kingdom");
        }
    }

    public static void main(String[] args) {
        new FlagButtonDemo();
    }
}
```

Q6.

a. Explain the event handling mechanism used by Swing and also develop the program. 10 Marks

Ans: **Event Handling**

Event handling is the mechanism that allows a Swing application to respond to user actions, such as clicking a button, typing in a text field, or moving the mouse. In a GUI, nothing happens automatically—event handling connects user interactions to the code that makes the app dynamic. Swing inherits its event-handling system from AWT but enhances it with additional features for its lightweight components. This section explains how it works, step-by-step, with examples.

What is an Event?

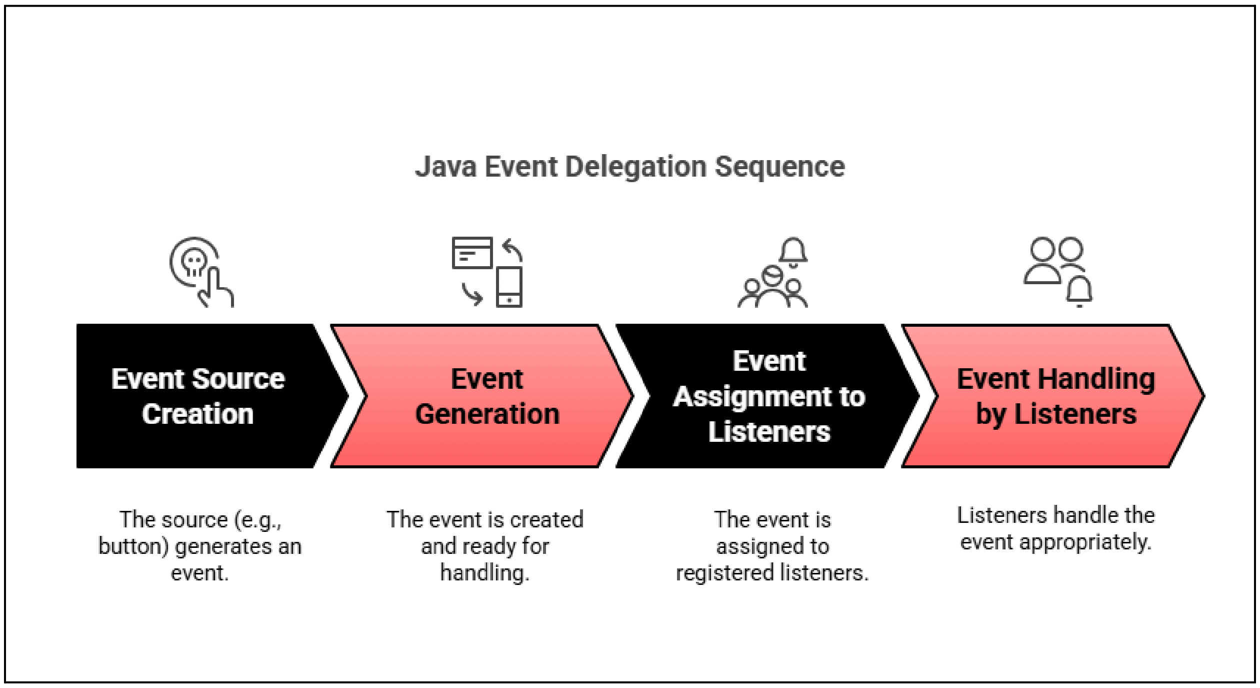
An event is something that happens in the app, usually triggered by the user (e.g., a button click) or the system (e.g., a window closing). **In Java, events are represented as objects, like ActionEvent for button clicks or MouseEvent for mouse actions.** Event handling is about detecting these events and running code in response.

Core Concepts

1. Event Source: The object where the event occurs (e.g., a JButton).
2. Event Listener: An object that "listens" for events and defines what to do when they happen.
3. Event Object: Carries details about the event (e.g., which button was clicked).
4. Event Registration: Connecting the listener to the source so it gets notified.

Swing uses AWT's event-handling framework (from java.awt.event) and adds Swing-specific events (from javax.swing.event) for advanced components.

How Event Handling Works in Swing
1. An event occurs (e.g., user clicks a JButton).
2. The event source (JButton) creates an event object (e.g., ActionEvent).
3. The event is sent to all registered listeners.
4. The listener's code runs, responding to the event (e.g., updating a label).


Java Event Delegation Sequence
Event Source Creation
Event Generation
Event Assignment to Listeners
Event Handling by Listeners
The source (e.g., button) generates an event.
The event is created and ready for handling.
The event is assigned to registered listeners.
Listeners handle the event appropriately.

Swing components have methods like addActionListener to register listeners, making it easy to hook up responses.

Button Click Event
Here's a simple Swing app with a button that prints a message when clicked:

```
import javax.swing.*;
import java.awt.event.*;

public class ButtonEventDemo {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Button Event Demo");
      frame.setSize(300, 200);
```

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton button = new JButton("Click Me");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button clicked!");
            }
        });

        frame.add(button);
        frame.setVisible(true);
    }
}
```

b. Illustrate the use of Radio buttons and also develop the program. 10 Marks

Radio Buttons are used when the user is allowed to **choose only one option** from a group. In Java Swing, the JRadioButton class is used for creating radio buttons, and the ButtonGroup class ensures mutual exclusivity — only one radio button in the group can be selected at a time.

Java Program: Radio Button Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class RadioButtonExample extends JFrame implements ActionListener {
    JRadioButton redButton, greenButton, blueButton;
    ButtonGroup colorGroup;
    JLabel resultLabel;

    public RadioButtonExample() {
        setTitle("Radio Button Example");
        setSize(300, 200);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        // Create radio buttons
        redButton = new JRadioButton("Red");
        greenButton = new JRadioButton("Green");
        blueButton = new JRadioButton("Blue");

        // Group them so only one can be selected
        colorGroup = new ButtonGroup();
        colorGroup.add(redButton);
        colorGroup.add(greenButton);
        colorGroup.add(blueButton);

        // Add action listeners
        redButton.addActionListener(this);
        greenButton.addActionListener(this);
        blueButton.addActionListener(this);

        // Label to display result
        resultLabel = new JLabel("Select your favorite color");

        // Add components to frame
        add(redButton);
        add(greenButton);
        add(blueButton);
        add(resultLabel);

        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (redButton.isSelected()) {
            resultLabel.setText("You selected: Red");
        } else if (greenButton.isSelected()) {
            resultLabel.setText("You selected: Green");
        } else if (blueButton.isSelected()) {
            resultLabel.setText("You selected: Blue");
        }
    }

    public static void main(String[] args) {
        new RadioButtonExample();
    }
}
```

Q7.

a. List and explain the core classes and interfaces in javax.servlet package 10 Marks

1. jakarta.servlet Package

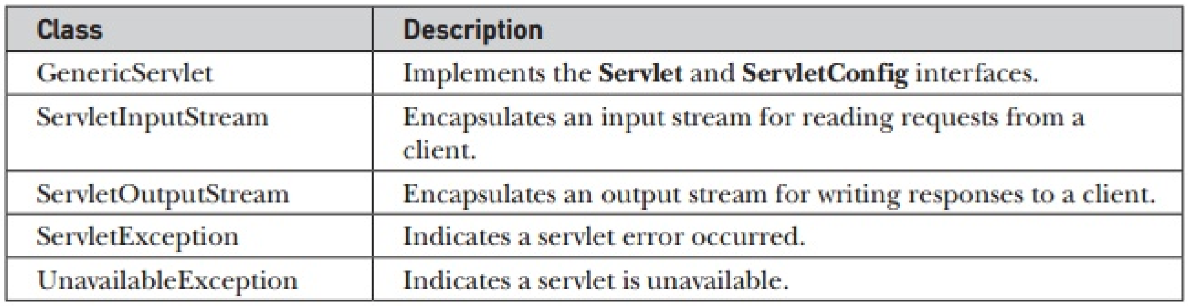
The jakarta.servlet package contains a number of interfaces and classes that establish the framework in which servlets operate.

The following table summarizes key interfaces

Interface	Description
Servlet	Defines methods that all servlets must implement.

ServletConfig	Enables a servlet to access its configuration information.
ServletContext	Enables a servlet to log events and access information about its environment.
ServletRequest	Provides client request information to a servlet.
ServletResponse	Assists a servlet in sending a response to the client.

The following table summarizes the core classes provided in the jakarta.servlet package:



Class	Description
GenericServlet	Implements the **Servlet** and **ServletConfig** interfaces.
ServletInputStream	Encapsulates an input stream for reading requests from a client.
ServletOutputStream	Encapsulates an output stream for writing responses to a client.
ServletException	Indicates a servlet error occurred.
UnavailableException	Indicates a servlet is unavailable.

The Servlet Interface

All servlets must implement the Servlet interface. It declares the init(), service() and destroy() methods that are called by the server during the life cycle of a servlet. A method is also provided that allows a servlet to obtain any initialization parameters. The method defined by Servlet are shown in the below table.

Method	Description
void init(ServletConfig config)	Called by the servlet container to initialize the servlet.
ServletConfig getServletConfig()	Returns a ServletConfig object containing the servlet's configuration and initialization parameters.
Void service(ServletRequest request, ServletResponse response)	Called by the servlet container to allow the servlet to respond to a request.
String getServletInfo()	Returns information about the servlet, such as author, version, and copyright.
void destroy()	Called by the servlet container to indicate that the servlet is being taken out of service and to allow it to clean up resources.

The ServletConfig Interface

The ServletConfig interface allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are summarized below.

Method	Description
String getInitParameter(String name)	Returns the value of the initialization parameter specified by name.
Enumeration<String> getInitParameterNames()	Returns an enumeration of all initialization parameter names.
ServletContext getServletContext()	Returns a reference to the ServletContext in which the servlet is running.
String getServletName()	Returns the name of the servlet instance.

The ServletContext Interface
The ServletContext interface enables servlets to obtain information about their environment.

Method	Description
Object getAttribute(String name)	Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.
String getMimeType(String file)	Returns the MIME type of the specified file, or null if the MIME type is not known.
String getRealPath(String path)	Returns a String containing the real path for a given virtual path.
String getServerInfo()	Returns the name and version of the servlet container on which the servlet is running.
void log(String msg)	Writes the specified message to a servlet log file, usually an event log.
void setAttribute(String name, Object object)	Binds an object to a given attribute name in this context.

The ServletRequest Interface
The ServletRequest interface enables a servlet to obtain information about a client request. Its methods are summarized in the table below.

Method	Description
Object getAttribute(String name)	Returns the value of a named attribute.

Enumeration<String> getAttributeNames()	Returns an enumeration of all attribute names in the request.
String getCharacterEncoding()	Returns the character encoding used for the request body.
int getContentLength()	Returns the length of the request body in bytes. *(Deprecated: better use getContentLengthLong())*
long getContentLengthLong()	Returns the length of the request body as a long.
String getContentType()	Returns the MIME type of the request body.
ServletInputStream getInputStream()	Returns a ServletInputStream to read binary data from the request body.
String getParameter(String name)	Returns the value of a request parameter as a String, or null if the parameter does not exist.
Enumeration<String> getParameterNames()	Returns an enumeration of parameter names.
String[] getParameterValues(String name)	Returns an array of String objects containing all the values for the specified parameter.
Map<String, String[]> getParameterMap()	Returns a Map of the request parameters.
String getProtocol()	Returns the name and version of the protocol used by the request (like HTTP/1.

The ServletResponse Interface
The ServletResponse interface enables a servlet to formulate a response for a client. Its methods are summarized in the table below.

Method	Description
String getCharacterEncoding()	Returns the name of the character encoding used in the response.
String getContentType()	Returns the MIME type of the response.
ServletOutputStream getOutputStream()	Returns a ServletOutputStream to write binary data to the response.

PrintWriter getWriter()	Returns a PrintWriter to send character text to the client.
void setCharacterEncoding(String charset)	Sets the character encoding for the response.
void setContentLength(int len)	Sets the length of the response body in bytes. *(Deprecated: better use setContentLengthLong())*
void setContentLengthLong(long length)	Sets the length of the response body as a long.
void setContentType(String type)	Sets the MIME type of the response.
void setBufferSize(int size)	Sets the preferred buffer size for the response.
int getBufferSize()	Returns the actual buffer size used for the response.
void flushBuffer()	Forces any content in the buffer to be written to the client.
void resetBuffer()	Clears the content of the buffer without clearing headers or status code.
boolean isCommitted()	Returns a boolean indicating if the response has been committed.
void reset()	Clears the buffer and resets the response.
void setLocale(Locale loc)	Sets the locale of the response.
Locale getLocale()	Returns the locale assigned to the response.

Classes of jakarta.servlet package
The GenericServlet Class
The GenericServlet class provides implementations of the basic life cycle methods for a servlet. GenericServlet implements the Servlet and ServletConfig interfaces. A method to append a string to the server log file is available. The signatures of this method -
Void log (String s)
Void log(String s, Throwable e)
S is the string to be appended to the log and e is the exception that occurred.

The ServletInputStream Class
The ServletOutputStream Class
The Servlet Exception Class

Example: Using jakarta.servlet

```
import jakarta.servlet.*;
import java.io.*;

public class BasicServlet implements Servlet {
   private ServletConfig config;

   public void init(ServletConfig config) throws ServletException {
      this.config = config;
      System.out.println("Initialized with " + config.getInitParameter("message"));
   }

   public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
      response.setContentType("text/plain");
      PrintWriter out = response.getWriter();
      out.println("Hello from jakarta.servlet");
   }

   public void destroy() {
      System.out.println("Servlet destroyed");
   }

   public ServletConfig getServletConfig() {
      return config;
   }

   public String getServletInfo() {
      return "Basic Servlet";
   }
}
```

Explanation
- import jakarta.servlet.*: Accesses Servlet, ServletRequest, etc.
- implements Servlet: Directly uses the interface.
- init(ServletConfig config): Stores config, reads an init parameter (e.g., "message").
- service(ServletRequest req, ServletResponse res): Sends a plain text response.
- destroy(): Logs shutdown.
- getServletConfig() and getServletInfo(): Required by the interface.

Deployment
1. Compile: javac -cp "C:\tomcat\lib\servlet-api.jar" BasicServlet.java
2. Place BasicServlet.class in tomcat/webapps/myapp/WEB-INF/classes.
3. Configure web.xml:

```
<web-app>
   <servlet>
```

```
        <servlet-name>BasicServlet</servlet-name>
        <servlet-class>BasicServlet</servlet-class>
        <init-param>
            <param-name>message</param-name>
            <param-value>Welcome</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>BasicServlet</servlet-name>
        <url-pattern>/basic</url-pattern>
    </servlet-mapping>
</web-app>
```

4. Start Tomcat, visit http://localhost:8080/myapp/basic.
- Speculative Output: Browser shows "Hello from jakarta.servlet"; console logs "Initialized with Welcome" and "Servlet destroyed" on shutdown.

Using GenericServlet

```
import jakarta.servlet.*;
import java.io.*;

public class GenericSimpleServlet extends GenericServlet {
    public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.println("Hello from GenericServlet");
    }
}
```

- Extends GenericServlet, only overriding service().
- init() and destroy() use default implementations.
- Output: "Hello from GenericServlet".

b. Develop a Java Servlet program to accept two parameters from a Webpage, find the sum of them and display the result in the webpage. Also give necessary HTML script to create Webpage. 10 Marks

1. HTML Page (Input Form)

Save this file as `sum.html` and place it in your web application's root directory (like `webapp` or `public_html`).

```
<!DOCTYPE html>
<html>
<head>
   <title>Sum Calculator</title>
</head>
<body>
   <h2>Enter Two Numbers</h2>
   <form action="SumServlet" method="post">
      Number 1: <input type="text" name="num1"><br><br>
      Number 2: <input type="text" name="num2"><br><br>
      <input type="submit" value="Calculate Sum">
   </form>
</body>
</html>
```

2. Java Servlet Code

Save this Java class as `SumServlet.java` in your `src` folder. The servlet should be mapped in `web.xml` or annotated.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;

@WebServlet("/SumServlet")
public class SumServlet extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {

      // Set response type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      try {
         // Read input parameters
         int num1 = Integer.parseInt(request.getParameter("num1"));
         int num2 = Integer.parseInt(request.getParameter("num2"));

         int sum = num1 + num2;

         // Output the result
         out.println("<html><body>");
         out.println("<h2>Sum Calculation Result</h2>");
         out.println("<p>Number 1: " + num1 + "</p>");
         out.println("<p>Number 2: " + num2 + "</p>");
         out.println("<p><strong>Sum = " + sum + "</strong></p>");
         out.println("</body></html>");
      } catch (NumberFormatException e) {
         out.println("<html><body><p style='color:red;'>Invalid input. Please enter valid
numbers.</p></body></html>");
      }
   }
}
```

Q8.

a. Define JSP. Explain different JSP tags with suitable example programs. 8 Marks

- A JSP is simpler to create than a Java servlet because a JSP is written in HTML rather than with the Java programming language.
- There are three methods that are automatically called when a JSP is requested and when the JSP terminates normally.
- jspInt()- method is called first when the JSP is requested and is used to initialize objects and variables that are used throughout the life of the JSP
- jspDestToy() - method is automatically called when the JSP terminates normally.
- service()- method is automatically called and retrieves a connection to HTTP

JSP Tags:

- **Comment tag:** A comment tag opens with, and is followed by a comment that usually describes the functionality of statements that follow the comment tag.
- **Declaration statement tag:** Declaration Tag (<%! ... %>) Used for declaring variables or methods.
- **Directive tags:** Directive tags are processed at translation time (before the page is executed) and set properties for the entire page or for the entire JSP file.
    - Types of Directive Tags:
        - Page Directive (<%@ page %>)
        - Include Directive (<%@ include %>)
        - Taglib Directive (<%@ taglib %>)

- **Expression tags:** The expression tag evaluates a Java expression and outputs the result directly into the HTML response sent to the client.

- **The scriptlet tag:** lets you write regular Java code inside a JSP file. This code is executed on the server when the JSP is processed.

1. Directive Tags

These provide global information about the JSP page.

Syntax:

```
<%@ directive attribute="value" %>
```

Common Directive: page

Example:

```
<%@ page contentType="text/html" language="java" %>
<html>
  <body>
    <h2>Welcome to JSP</h2>
  </body>
</html>
```

---

2. Declaration Tags

Used to declare variables and methods that can be used throughout the JSP page.

Syntax:

```
<%! declaration %>
```

Example:

```
<%@ page contentType="text/html" %>
<%!
    int counter = 0;
    public int incrementCounter() {
        return ++counter;
    }
%>
<html>
  <body>
    Counter: <%= incrementCounter() %>
  </body>
</html>
```

---

3. Scriptlet Tags

Used to write Java code inside JSP pages. The code is executed each time the page is requested.

Syntax:

```
<% java code %>
```

Example:

```
<%@ page contentType="text/html" %>
<html>
  <body>
    <%
      int a = 5;
      int b = 10;
      int sum = a + b;
    %>
    <p>Sum = <%= sum %></p>
  </body>
</html>
```

---

4. Expression Tags

Used to output data directly to the HTML. It evaluates and displays the result.

Syntax:

```
<%= expression %>
```

Example:

```
<%@ page contentType="text/html" %>
<html>
  <body>
    <p>Today's year: <%= java.time.Year.now() %></p>
  </body>
</html>
```

---

5. Comment Tags

Used to add comments that are **not sent to the client browser**.

Syntax:

```
<%-- comment --%>
```

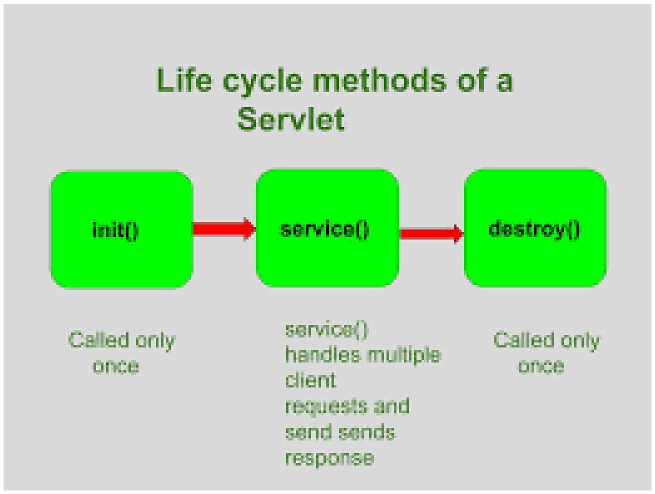
Example:

```
<%-- This is a JSP comment --%>
```

b. Explain the life cycle of Servlet. 4 Marks

The Life Cycle of a Servlet

The life cycle of a servlet describes the stages it goes through from creation to destruction, managed by a **web container (like Apache Tomcat).** Unlike standalone Java programs, servlets don't run independently—they're controlled by the container, which handles their instantiation, request processing, and cleanup. Understanding this life cycle is key to writing effective servlets, as it dictates when and how your code executes. **The life cycle has three main phases: initialization, service, and destruction, each tied to specific methods in the Servlet interface.**


Life cycle methods of a Servlet
init()
service()
destroy()
Called only once
service() handles multiple client requests and send sends response
Called only once

Overview of the Life Cycle
1. **Initialization**: The servlet is created and set up (happens once).
2. **Service**: The servlet handles client requests (happens multiple times).
3. **Destruction**: The servlet is shut down and cleaned up (happens once).

These phases are triggered by the container based on server events, like starting up, receiving requests, or shutting down.

The Servlet Interface
The life cycle is defined by the **jakarta.servlet.Servlet interface**, which all servlets implement (directly or via classes like HttpServlet). It includes five methods, but three are central to the life cycle:
- **init(ServletConfig config):** Called once when the servlet is initialized.
- **service(ServletRequest req, ServletResponse res):** Called for each client request.
- **destroy():** Called once when the servlet is removed.

The other two methods—**getServletConfig() and getServletInfo()—provide metadata but aren't part of the core life cycle.**

Detailed Phases

1. Initialization (init Method)
- When: The container loads the servlet, either at server startup or the first request (configurable).
- Purpose: Sets up the servlet before it handles requests—e.g., initializing variables, database connections, or resources.
- How: The container:
  1. Instantiates the servlet (calls new MyServlet()).
  2. Calls init(ServletConfig config) with a ServletConfig object containing configuration data (e.g., from web.xml).
- Key Points:
  - Runs only once per servlet instance.
  - The servlet isn't ready for requests until init completes.
  - You override init() to add setup code.

2. Service (service Method)
- When: Every time a client request arrives (e.g., a browser sends GET or POST).
- Purpose: Processes the request and generates a response—e.g., sending HTML to the browser.
- How: The container:
  1. Receives a request (e.g., http://localhost:8080/myServlet).
  2. Calls service(ServletRequest req, ServletResponse res) with request and response objects.
- Key Points:
  - Runs multiple times—once per request.
  - For HttpServlet, service() delegates to methods like doGet() or doPost() based on the HTTP method.
  - Thread-safe: One servlet instance handles multiple requests via threads, so instance variables need caution.

3. Destruction (destroy Method)
- When: The container removes the servlet, usually at server shutdown or if the servlet is reloaded.
- Purpose: Cleans up resources—e.g., closing database connections or files.
- How: The container calls destroy() after all requests are handled and before the servlet is garbage-collected.
- Key Points:
  - Runs only once.
  - No requests are processed after this.
  - You override destroy() for cleanup.

Life Cycle in Action
1. Tomcat starts, loads LifeCycleServlet, calls init().
2. User requests /lifecycle, service() (or doGet()) runs, sends HTML.
3. More requests repeat step 2.
4. Tomcat shuts down, calls destroy().

c. What is a cookie? List out the method defined by cookie. Develop a Java program to add a cookie. 8 Marks

Cookies are name-value pairs (e.g., "username=Alice") sent from a server to a client as part of an HTTP response. The client (browser) stores them and includes them in future requests to the same server. They're part of the HTTP protocol and handled in servlets via the Cookie class and HttpServletResponse/HttpServletRequest objects.

Key Features:
- Client-Side Storage: Stored in the browser, not the server.
- Persistent or Temporary: Can last for a session or a set time.
- Small Size: Limited to 4KB per cookie, with browser limits on total cookies.

Uses:
- Tracking: Count visits or remember user choices.
- Authentication: Store login tokens.
- Personalization: Save preferences (e.g., theme).

The Cookie Class
The jakarta.servlet.http.Cookie class represents a cookie:
- Constructor: Cookie(String name, String value) creates a cookie.
- **Methods:**
  - **getName():** Returns the cookie's name.
  - **getValue():** Returns the cookie's value.
  - **setMaxAge(int seconds):** Sets how long the cookie persists (in seconds; -1 = session-only, 0 = delete).
  - **setPath(String path):** Sets the URL path the cookie applies to (e.g., "/myapp").

Setting Cookies
Cookies are added to the response using HttpServletResponse:
- addCookie(Cookie cookie): Sends the cookie to the client.

Reading Cookies
Cookies are retrieved from the request using HttpServletRequest:
- getCookies(): Returns an array of Cookie objects sent by the client (null if none).

Example 1: Setting a Cookie

A servlet that sets a cookie:

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import java.io.*;

public class SetCookieServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        Cookie cookie = new Cookie("user", "Alice");
        cookie.setMaxAge(3600); // 1 hour
        response.addCookie(cookie);

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("Cookie 'user' set to 'Alice'");
        out.println("</body></html>");
    }
}
```

Explanation
- Cookie cookie = new Cookie("user", "Alice"): Creates a cookie named "user" with value "Alice".
- setMaxAge(3600): Makes the cookie persist for 1 hour (3600 seconds).
- response.addCookie(cookie): Adds the cookie to the response.
- out.println(...): Confirms the action.

Deployment
1. Compile: javac -cp "C:\tomcat\lib\servlet-api.jar" SetCookieServlet.java
2. Place in tomcat/webapps/myapp/WEB-INF/classes.
3. Web.xml:

```
<web-app>
   <servlet>
      <servlet-name>SetCookieServlet</servlet-name>
      <servlet-class>SetCookieServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>SetCookieServlet</servlet-name>
      <url-pattern>/setcookie</url-pattern>
   </servlet-mapping>
</web-app>
```

4. Test: http://localhost:8080/myapp/setcookie

Output
- Browser shows: "Cookie 'user' set to 'Alice'".
- Browser's developer tools (e.g., Chrome DevTools > Application > Cookies) show "user=Alice" with a 1-hour expiration.

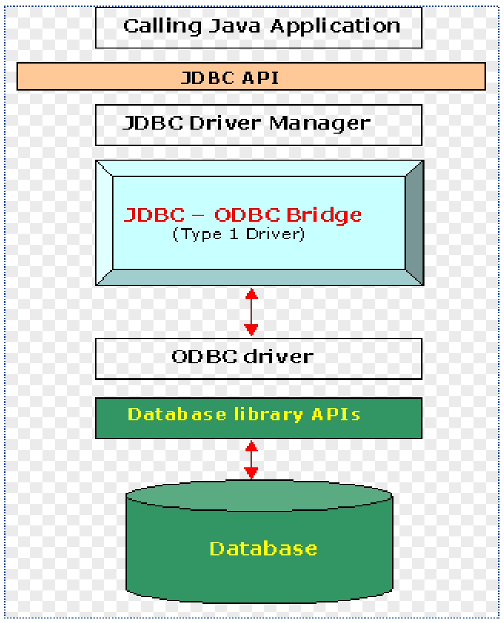
Q9.

a. Explain 4 types of JDBC drivers. 5 Marks

Type 1 JDBC-to-ODBC Driver

- ODBC (Open Database Connectivity)
    - Created by Microsoft
    - The basis on which Sun Microsystems, Inc. created JDBC
    - A DBMS-independent database program. The JDBC-to-ODBC Driver

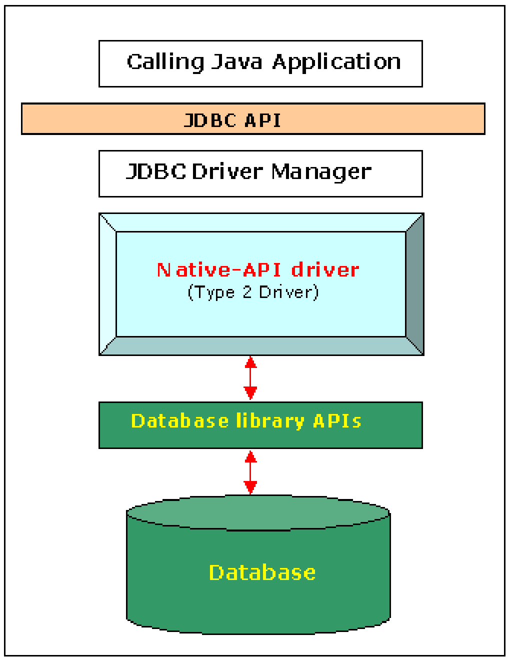
    - Also known as JDBC-ODBC bridge
    - Used to translate DBMS calls between the JDBC specification and the ODBC specification
        - Receives a message from the J2EE component that conforms to the JDBC specification.
        - Then translates this into a message format understood by the DBMS.
- Note: Avoid using mission-critical systems because the extra translation might negatively impact performance.



<u>Type 2 Java/Native Code Driver</u>

- Uses native (platform-specific) database client libraries.
- JDBC calls are converted into native calls.

- Disadvantage
    - Loss of **portability**
    - **Platform dependent**



Type 3 JDBC Driver

- Also known as the Java Protocol
- Uses a middleware (server) to translate JDBC calls into database-specific calls.
- Converts SQL queries into JDBC-formatted statements.
- The JDBC formatted statements are translated into the format required by the DBMS.
- Platform independent.

Type 4 JDBC Driver

- Also known as the Type 4 database protocol.
- Like Type 3 except
  - SQL queries are translated into the format required by the DBMS
  - It does not need to be converted to JDBC format system.
- The fastest way to communicate SQL queries to DBMS as it does not have the overhead of conversion of calls to other API's.
- Pure Java implementation that communicates directly with the database over the network.

b. Construct a code snippet to describe the various steps involved in JDBC process. 10 Marks

c. Various Steps Involved in JDBC Process

1. **Loading** the JDBC driver
2. **Connecting** to the DBS
3. **Creating and executing** a statement
4. **Processing data** returned by the DBMS
5. **Terminating** the *connection* with the DBMS

1. <u>Loading the JDBC driver</u>

- Class.forName() is used to load J2EE component
- We are going to develop an application that uses Microsoft Access
- Then we must write a routing that loads the JDBC/ODBC Bridge driver called sun.jdbc.odbc. JdbcOdbcDriver.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver)
```

- If you had used a MySQL driver

```
try{
      Class.forName("com.mysql.cj.jdbc.Driver");

        }catch(ClassNotFoundException err) {
              System.out.println("Unable to connect:"+err);
          System.exit(1);
                }
```

2. <u>Connecting to the DBMS</u>

- Connect using DriverManager.getConnection() method.
- java.sql.DriverManager class is the higher class in the java.sql hierarchy and is responsible for managing driver information.
- Parameters passed
    - o URL: a string object that contains the driver name and the name of the database being accessed.
    - o Username (optional)
    - o Password (optional)
- Returns a Connection interface that is used to reference the database.

```
String url = “jdbc:odbc:CustomerInformation”;
String userID = “jim”;
String password = “pswd”;
private Connection Db;
try{
    Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
    Db = DriverManager.getConnection (url, userID, password);
}
```

- If you had used a MySQL driver

```
try {
        Connection db=
DriverManager.getConnection("jdbc:mysql://localhost:3306/cust_db","root","");
                System.out.println("Successfully connected to database...");
        }catch(SQLException e) {
                System.out.println("Connection not Successful");
                e.printStackTrace();
        }
```

3. <u>Create and Execute an SQL Statement.</u>

- After the connection is made, SQL query can be sent to the DBMS for processing
- SQL query – consists of a series of SQL commands that direct the DBMS.
- The createStatement () method of the Connection interface is used to create a Statement object.
- The Statement object is then used to execute a query that returns a ResultSet object that contains a response from the DBMS.

```
Statement stmt;
ResultSet rs;
try{
    String query = “SELECT * FROM Customers”;
    stmt = Db.createStatement();
    rs = stmt.executeQuery(query);
    stmt.close();
}
…
```

4. **Process Data Returned by the DBMS**

- The java.sql.ResultSet object is assigned the results received from the DBMS after the query is processed.
- next() method of the ResultSet class
  - The first time next() method is called, positions ResultSet pointer at the first row of the ResultSet.
  - Returns a Boolean value that, if false, indicates that no rows are present in the ResultSet.
  - A true value means at least one row is present in the ResultSet.
- do…while() loop is used to loop through each record
- getString() method – used to copy the column contents in the ResultSet
  - can pass the column name or the column number.

```
ResultSet rs;
String fname, lname;
boolean records = rs.next();
if(!records)
        System.out.println("No data returned");
else {
  {
        do{
           fname = rs.getString("FirstName");
           lname = rs.getString("LastName");
           System.out.println(fname + " " + lname);
        }while(rs.next() );
 }
```

5. **<u>Terminate the connection to the database</u>**

- Use the close() method of the Connection object to terminate the connection to the DBMX.
- Throws an exception if a problem is encountered.

```
Db.close();
```

d. Explain database metadata and ResultSet metadata. 5 Marks

MetaData

- data about data
- two types

    - DatabaseMetaData interface
    - ResultSet MetaData
- a J2EE component can access metadata by using the DatabaseMetaData interface.
- used to retrieve information about db, table, columns and indexes

DatabaseMetaData interface.
(https://docs.oracle.com/javase/7/docs/api/java/sql/DatabaseMetaData.html)

- String getDatabaseProductName() – returns the product name of the database
- String getUserName() – returns the username
- String getURL() – returns the URL of the database
- ResultSet getSchemas() – returns all the schema names available in this database
    - TABLE_SCHEM
    - TABLE_CATALOG
- ResultSet getPrimaryKeys(String catalog, String schema, String table) – returns primary keys
    - give “” for catalog and schema if table name is sufficient
    - ResultSet : each row is a primary key column description.
- getProcedures() – returns stored procedures name
- getTables() – returns names of tables in the database
    - ResultSet getTables(**String** catalog, **String** schemaPattern, **String** tableNamePattern, **String**[] types)
    - getTables(null,null, “customers”,null)

ResultSet Metadata
(https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSetMetaData.html)
retrieved by calling the getMetaData() method of the ResultSet Object.

ResultSetMetaData rsm = rs.getMetaData();

- int getColumnCount() – returns the number of columns contained in the ResultSet
- String getCoulmnName(int number) – returns the name of the column specified by the column number
- int getCoulmnType(int number) – returns the data type of the column specified by the column number.
    - https://docs.oracle.com/javase/7/docs/api/java/sql/Types.html
    - BIGINT, CHAR, CLOB,…

Q10.

a. What is a statement object in JDBC? Explain the following statement objects 10 Marks
    i. Callable statement objects
    ii. Prepared Statement objects

Statement Objects

- J2EE component creates and sends a query to access data in database.
- 3 types of statement objects are used to execute the query
    - The **Statement** Object
    - **PreparedStatement** Object

- o **CallableStatement** Object

1. The Statement Object

- used whenever a J2EE component needs to immediately execute a query without having the query compiled.
  - o executeQuery()
    - returns ResultSet object containing rows, columns and metadata
  - o execute()
    - when multiple results are returned.
  - o executeUpdate()
    - used to execute queries that contain UPDATE and DELETE SQL statements.
    - returns an integer indicating a number of rows updated.
    - also used for the INSERT statement

<u>Code using executeQuery()</u>

```
….
Connection db;
Statement stmt;
ResultSet rs;
try{
        String query = “SELECT * FROM customers”;
        stmt = db.createStatement();
        rs = stmt.excecuteQuery(query);
        //…….
        stmt.close();
}catch(SQLException e) {
        System.err.println(“SQL Error”  + err);
}
```

<u>Code using executeUpdate() to update</u>

```
….
Connection db;
Statement stmt;
int numRowsUpdated;
try{
        String query = “UPDATE customers SET PAID= ‘Y’ WHERE BALANCE = 0”;
        stmt = db.createStatement();
        numRowsUpdated= stmt.excecuteUpdate(query);
        //…….
        stmt.close();
}catch(SQLException e) {
        System.err.println(“SQL Error”  + err);
}
```

2. PreparedStatement Object

- An SQL query must be compiled before the DBMS processes the query
- Compiling occurs after the Statement objects' execution method.
- Compiling a query is an acceptable overhead if the query is called once.

- compiling several times becomes an expensive overhead
- Solution: PreparedStatement Object
    - **precompiles** and executes an SQL statement, known as **late binding**
    - a? placeholder is used in the query to later insert the value for the query.

Steps to use PreparedStatement Object

1. prepareStatement() method of the Connection object is called to return the PreparedStatement
    - prepareStatement() method is passed the query.
2. setxxx() method is used to replace the ? placeholder.
    - xxx – refers to a data type, eg., setString()
        - 2 parameters are passed -
            - The first parameter: identifies the position of the ? placeholder
            - Second parameter: the late binding variable.
3. the executeQuery() method of the PreparedStatement object is called.
    - it doesn't require a parameter 'cos query is already associated with the PreparedStatement object.

Code using PreparedStatement

```
Connection db;
ResultSet rs;
….
try {
        String qPrep ="Select * from customers where custNumber = ?";
        PreparedStatement pstmt =db.prepareStatement(qPrep);
        pstmt.setString(1, "C0100");
        ResultSet prs = pstmt.executeQuery();
        prs.next();
        System.out.println("\n**Customer with Number : C0100 is ");
        System.out.println(prs.getString(3)+" "+prs.getString(4));
 }catch(SQLException err) {
        err.printStackTrace();
 }
```

3. CallableStatement Object

- used to call a stored procedure from within a J2EE object.
    - stored procedure: a block of code identified by a unique name, executed by invoking the name of the stored procedure.
- uses 3 types of parameters
    - IN:
        - contains data that needs to be passed to a stored procedure.
        - value is assigned using setxxx() method
    - OUT
        - contains value returned by the stored procedure
        - must be registered using the registerOutParameter() method.
        - later received using getxxx() method

- o INOUT
    - ▪ used to pass and retrieve information from a stored procedure.

Steps to use CallableStatement Object

1. **prepareCall()** of the Connection object is called and passed the query
    - returns a CallableStatement object.
2. **registerOutParameter()** has two arguments
    - 1st parameter: represents the number of the parameter in the stored procedure
    - 2nd parameter: data type of the value returned by the stored procedure, eg. VARCHAR.
3. **execute()** method of CallableStatement object is used to execute the query
    - o need not be passed as it is already identified in the prepareCall() method.

```
delimiter; $$

create procedure LastOrderNumber(IN var1 int, OUT firstName CHAR(20))
   -> BEGIN
   -> select fname INTO firstName from customers where ID = var1;
   -> END$$

call LastOrderNumber(1, @L);
Query OK, 1 row affected (0.05 sec)

mysql> select @L;
```

__**Code to use CallableStatement Object.**__

```
Connection db;
ResultSet rs;
….
 /*  CallableStatement Demo */
 try {
          String cQry = "CALL LastOrderNumber(1,?)";
         CallableStatement cstmt = db.prepareCall(cQry);
         cstmt.registerOutParameter(1, Types.VARCHAR);
         cstmt.execute();
         String fname = cstmt.getString(1);
         System.out.println("\n**Call Statement Demo");
         System.out.println(fname);

 }catch(SQLException err) {
          err.printStackTrace();
 }
```

b. Develop a Java program to execute a database transaction. 6 Marks

Java Program: Transaction Example using JDBC

Requirements:

* Java installed
* MySQL or any RDBMS (you can adapt connection URL accordingly)
* JDBC driver in classpath
* A sample database with table `accounts` having `id` and `balance`

```java
import java.sql.*;

public class BankTransaction {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/bankdb";  // Replace with your DB
        String user = "root";                               // Your DB username
        String password = "password";                       // Your DB password

        Connection conn = null;

        try {
            conn = DriverManager.getConnection(url, user, password);
            conn.setAutoCommit(false);  // Start transaction

            // Withdraw from Alice
            PreparedStatement withdraw = conn.prepareStatement(
                "UPDATE accounts SET balance = balance - ? WHERE id = ?"
            );
            withdraw.setDouble(1, 1000);  // Amount
            withdraw.setInt(2, 1);        // Alice's ID
            withdraw.executeUpdate();

            // Deposit into Bob
            PreparedStatement deposit = conn.prepareStatement(
                "UPDATE accounts SET balance = balance + ? WHERE id = ?"
            );
            deposit.setDouble(1, 1000);   // Amount
            deposit.setInt(2, 2);         // Bob's ID
            deposit.executeUpdate();

            conn.commit();  // Commit transaction
            System.out.println("Transaction successful.");

        } catch (SQLException e) {
            try {
                if (conn != null) {
                    conn.rollback();  // Rollback on error
                    System.out.println("Transaction failed. Rolled back.");
            }
        } catch (SQLException rollbackEx) {
            rollbackEx.printStackTrace();
        }
        e.printStackTrace();
    } finally {
        try {
            if (conn != null)
                conn.close();  // Close connection
        } catch (SQLException closeEx) {
            closeEx.printStackTrace();
        }
    }
  }
}
```

c. Show any two syntax of establishing a connection to database. 4 Marks

The **two common syntaxes** to establish a connection to a database in Java using JDBC:

1. Using DriverManager.getConnection() with individual parameters

```
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/mydatabase", "username", "password");
```

- **Driver:** MySQL (com.mysql.cj.jdbc.Driver)
- **URL format:** jdbc:mysql://<host>:<port>/<database>
- **Example:** Connects to mydatabase at localhost on port 3306.

---

2. Using Properties object for more flexible configuration

```
Properties props = new Properties();
props.setProperty("user", "username");
props.setProperty("password", "password");

Connection con = DriverManager.getConnection(
    "jdbc:postgresql://localhost:5432/mydatabase", props);
```

- This format is useful when passing additional settings like SSL, encoding, etc.
- **Driver:** PostgreSQL (org.postgresql.Driver)