www.ignousite.com
Course Code : MCS-024
Course Title : Object Oriented Technologies and Java Programming
Assignment Number : BCA/MCA/PGDCA/024/Assignment/2022-23
Maximum Marks : 100 www.ignousite.com
Last Date of Submission : 31st October, 2022 (for July session)
: 15th April, 2023 (for January session)

Q1. (a) What is Object Oriented Programming? Explain concept of encapsulation with example in java.

Ans. Object Oriented Programming: The object-oriented programming is basically a computer programming design philosophy or methodology that organizes/ models software design around data, or objects rather than functions and logic. An object is referred to as a data field that has unique attributes and behavior. Everything in OOP is grouped as self-sustainable objects.

It is the most popular programming model among developers. It is well suited for programs that are large, complex, and actively updated or maintained. It simplifies software development and maintenance by providing major concepts such as abstraction, inheritance, polymorphism, and encapsulation. These core concepts support OOP.

A real-world example of OOP is the automobile. It more completely illustrates the power of object-oriented design.

Concept of encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, that it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of its own class in which it is declared.
- As in encapsulation, the data in a class is hidden from other classes using the data hiding concept which is achieved by making the members or methods of a class private, and the class is exposed to the end-user or the world without providing any details behind implementation using the abstraction concept, so it is also known as a combination of data-hiding and abstraction.
- Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.
- It is more defined with the setter and getter method.

Example: A capsule which is mixed of several medicines. We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The Java Bean class is the example of a fully encapsulated class.

(b) Explain use of different operators available in java.

Ans. Java Operator: Java operators are symbols that are used to perform operations on variables and manipulate the values of the operands. Each operator performs specific operations. Let us consider an expression 5 + 1 = 6; here, 5 and 1 are operands, and the symbol + (plus) is called the operator. We will also learn about operator precedence and operator associativity.

Operator Type	Category	Precedence
Unary	postfix	expr++ expr--
	prefix	++expr --expr +expr -expr ~ !
Arithmetic	multiplicative	* / %
	additive	+ -

Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Q2. (a) What is a class? Explain how you will define Book class in java. Also, explain use of getter and setter methods.

Ans. Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

Book class in java: Java provides a reserved keyword class to define a class. The keyword must be followed by the class name. Inside the class, we declare methods and variables.

```
class Book{

  String title;
  String author;
  double price;
  int option;

  //constructor
  Book(String title, String author, double newPrice){
    title = title;
    author = author;
    price = newPrice;

  }

  public String getTitle(){
    return title;
  }
  public String getAuthor(){
    return author;
  }
  public double getPrice(){
    return price;
  }
  public int getOption(){
    return option;
  }

  public void setPrice(int option, double price, double newPrice){
```

```
    if (option == 1){
      price = 20.00;
      newPrice = price;
    }
    else if (option == 2){
      price = 15.00;
      newPrice = price;
    }
    else if (option == 3){
      price = 23.90;
      newPrice = price;
    }
    else if (option == 4){
      price = 27.30;
      newPrice = price;
    }
    else if (option == 5){
      price = 50.00;
      newPrice = price;
    }
    else if (option == 6){
      price = 13.50;
      newPrice = price;
    }
  }
  public void setOption(int newOption){
    option = newOption;
  }
}
```

Use of getter and setter methods: Getter and setter methods are frequently used in Java programming. Getter and setter methods in Java are widely used to access and manipulate the values of class fields. Usually, class fields are decorated with a private access specifier. Thus, to access them, public access specifiers are used with the getter and setter methods.

One may argue that declare the class fields as public and remove the getter and setter methods. However, such a coding style is bad, and one may put some absurd value on the class fields. Let's understand it with the help of an example.

```
public class GetterSetterExample
{
  public salary;

  public storeSalaryDB(int salary)
  {
    // code for storing the salary in the database
  }

  // main method
```

```
    public static void main(String argvs[])
    {
     GetterSetterExample obj = new GetterSetterExample();

     obj.salary = -50000;

     // storing salary in database
     obj.storeSalaryDB(salary);

    }
}
```

(b) Explain use of super and final keywords in java.

Ans. Super Keyword: The super keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Final keyword: The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

(c) Write a java program to find the factorial of a given number.

Ans.

Program

```
import java.util.Scanner;
  public class Factorial {
    public static void main(String args[]){
      int i, factorial=1, number;
      System.out.println("Enter the number to which you need to find the factorial:");
      Scanner sc = new Scanner(System.in);
      number = sc.nextInt();

      for(i = 1; i<=number; i++) {
        factorial = factorial * i;
```

```
    }
    System.out.println("Factorial of the given number is: "+factorial);
  }
}
```

Output:

Enter the number to which you need to find the factorial:
25
Factorial of the given number is: 2076180480

Q3. Write a java program to create a Teacher class and define constructors for this class. Inherit Professor class, Associate_Professor class, and Assistant Professor class from the Teacher class. Define appropriate methods to calculate salary of teachers. Show how to implement method overriding in this program. Make necessary assumptions.

Ans.

```
import java.util.Scanner;
class Teacher {
int acno;
float bal=0;
Scanner get = new Scanner(System.in);
accounts()
{
System.out.println("Enter Account Number of Teacher:");
acno = get.nextInt();
System.out.println("Enter Initial Balance:");
bal = get.nextFloat();
}
void deposit()
{
float amount;
System.out.println("Enter Amount to be Deposited:");
amount = get.nextFloat();
bal = bal+amount;
System.out.println("Deposited! Account Balance is "+bal);
```

```
}
}
class savings extends accounts
{
//accounts a=new accounts();
//a.deposit();
void withdraw()
{
float amount;
System.out.println("Enter Amount to be Withdrawn:");
amount = get.nextFloat();
if(amount<bal)
{
bal = bal-amount;
System.out.println("Amount Withdrawn!! Available Balance: "+bal);
}
else
{
System.out.println("Insufficient funds!!");
}
}
}
public class Teacher {
public static void main(String[] args)
{
savings myObj = new savings();
myObj.deposit();
myObj.withdraw();
```

```
    }

    }
```

Output:

```
Enter Account Number of Teacher: 101
Enter Initial Balance: 500
Enter Amount to be Deposited: 100
Deposited! Account Balance is: 600
Enter Amount to be Withdrawn: 200
Amount Withdrawn!! Available Balance: 400
Insufficient funds!!
```

Q4. (a) Explain uses of abstract class in java with the help of an example.

Ans. Abstract class in Java: A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body). Before learning the Java abstract class, let's understand the abstraction in Java first.

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Example:

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Output:

```
running safely
```

(b) Explain accessibility rules for packages in java.

Ans. Accessibility means scope rules like where you can access or where you cannot access.

We have 4 access specifiers in java.

1.Public
2.Private
3.Protected
4.default

1.Public : We can access public variables from anywhere from any class. you can access freely, there is n o restrictions on this.

2. Private : This is most restricted access specifiers. We cannot access this from outside of the class, only we can access within the class.

3. Protected : This can be accessed in child class i,e if parent having the protected variable the child able to access that variable freely. It provides package level access.

4: Default : If you don't provide any access specifier, that comes under default. It provides only the package level access.

(c) What is polymorphism? Explain different types of polymorphism in java programming with the help of example.

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

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Types of polymorphism: There are two types of polymorphism in Java:

- Compile-time polymorphism
- Runtime polymorphism

Compile-time polymorphism: It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

Example:

```
// Java Program for Method overloading
// By using Different Types of Arguments

// Class 1
// Helper class
class Helper {

    // Method with 2 integer parameters
    static int Multiply(int a, int b)
    {

        // Returns product of integer numbers
        return a * b;
    }

    // Method 2
    // With same name but with 2 double parameters
```

```
    static double Multiply(double a, double b)
    {

        // Returns product of double numbers
        return a * b;
    }
}

// Class 2
// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Calling method by passing
        // input as in arguments
        System.out.println(Helper.Multiply(2, 4));
        System.out.println(Helper.Multiply(5.5, 6.3));
    }
}
```

Output:

```
8
34.65
```

Runtime polymorphism: It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Example:

```
// Java Program for Method Overriding

// Class 1
// Helper class
class Parent {

    // Method of parent class
    void Print()
    {

        // Print statement
        System.out.println("parent class");
    }
}
```

```
// Class 2
// Helper class
class subclass1 extends Parent {

        // Method
        void Print() { System.out.println("subclass1"); }
}

// Class 3
// Helper class
class subclass2 extends Parent {

        // Method
        void Print()
        {

                // Print statement
                System.out.println("subclass2");
        }
}

// Class 4
// Main class
class GFG {

        // Main driver method
        public static void main(String[] args)
        {

                // Creating object of class 1
                Parent a;

                // Now we will be calling print methods
                // inside main() method

                a = new subclass1();
                a.Print();

                a = new subclass2();
                a.Print();
        }
}
```

Output:

subclass1
subclass2

Q5. (a) What is interface? Explain difference between abstract class and interface with the help of examples. Write a java program to demonstrate use of interface.

Ans. Interface: An interface in Java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Abstract class: A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Example:

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Output:

running safely

Interface: An interface in Java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Example:

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Output:

Hello

A java program to demonstrate use of interface:

```
// Java program to demonstrate use of
// interface

import java.io.*;

// A simple interface
interface In1 {

        // public, static and final
        final int a = 10;

        // public and abstract
        void display();
}

// A class that implements the interface.
class TestClass implements In1 {

        // Implementing the capabilities of
        // interface.
        public void display(){
        System.out.println("Geek");
        }

        // Driver Code
        public static void main(String[] args)
        {
                TestClass t = new TestClass();
                t.display();
                System.out.println(a);
        }
}
```

Output:

```
Geek

10
```

(b) What is an exception? Explain various causes of exceptions. Explain exceptions hierarchy in java.

Ans. Exception: An exception is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

Various causes of exceptions: A java exception can be thrown only in the following three scenarios:

(1) An abnormal execution condition was synchronously detected by the Java virtual machine.
–When evaluation of an expression violates the normal semantics (Example: an integer divide by zero)
– An error occurs in loading or linking part of the program
– When limitation on a resource is exceeded (Example: using too much memory)

(2) A throw statement was executed.

(3) An asynchronous exception occurred.
– The stop method (deprecated) of class Thread was invoked
– An internal error has occurred in the java virtual machine

Exceptions hierarchy in java: The class at the top of the exception class hierarchy is the Throwable class, which is a direct subclass of the Object class. Throwable has two direct subclasses - Exception and Error.

The Exception class is used for exception conditions that the application may need to handle. Examples of exceptions include IllegalArgumentException, ClassNotFoundException and NullPointerException.

The Error class is used to indicate a more serious problem in the architecture and should not be handled in the application code. Examples of errors include InternalError, OutOfMemoryError and AssertionError.

Exceptions are further subdivided into checked (compile-time) and unchecked (run-time) exceptions. All subclasses of RuntimeException are unchecked exceptions, whereas all subclasses of Exception besides RuntimeException are checked exceptions.

Q6. (a) What is multithreading? What is thread priority? Describe interthread communications in java with the help of a program.

Ans. Multithreading: Multithreading in Java is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.
However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Thread priority: Priorities in threads is a concept where each thread is having a priority which in layman's language one can say every object is having priority here which is represented by numbers ranging from 1 to 10.

- The default priority is set to 5 as excepted.
- Minimum priority is set to 1.
- Maximum priority is set to 10.

Interthread communications in java: Inter-thread communication in Java is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.

Program:

```
// Java program to demonstrate inter-thread communication
// (wait(), join() and notify())

import java.util.Scanner;
```

```
public class threadexample
{
	public static void main(String[] args) throws InterruptedException
	{
		final PC pc = new PC();

		// Create a thread object that calls pc.produce()
		Thread t1 = new Thread(new Runnable()
		{
			@Override
			public void run()
			{
				try
				{
					pc.produce();
				}
				catch(InterruptedException e)
				{
					e.printStackTrace();
				}
			}
		});

		// Create another thread object that calls
		// pc.consume()
		Thread t2 = new Thread(new Runnable()
		{
			@Override
			public void run()
			{
				try
				{
					pc.consume();
				}
				catch(InterruptedException e)
				{
					e.printStackTrace();
				}
			}
		});

		// Start both threads
		t1.start();
		t2.start();
```

```
        // t1 finishes before t2
        t1.join();
        t2.join();
    }

    // PC (Produce Consumer) class with produce() and
    // consume() methods.
    public static class PC
    {
        // Prints a string and waits for consume()
        public void produce()throws InterruptedException
        {
            // synchronized block ensures only one thread
            // running at a time.
            synchronized(this)
            {
                System.out.println("producer thread running");

                // releases the lock on shared resource
                wait();

                // and waits till some other method invokes notify().
                System.out.println("Resumed");
            }
        }

        // Sleeps for some time and waits for a key press. After key
        // is pressed, it notifies produce().
        public void consume()throws InterruptedException
        {
            // this makes the produce thread to run first.
            Thread.sleep(1000);
            Scanner s = new Scanner(System.in);

            // synchronized block ensures only one thread
            // running at a time.
            synchronized(this)
            {
                System.out.println("Waiting for return key.");
                s.nextLine();
                System.out.println("Return key pressed");

                // notifies the produce thread that it
                // can wake up.
                notify();
```

```
                    // Sleep
                    Thread.sleep(2000);
                }
            }
        }
}
```

Output:

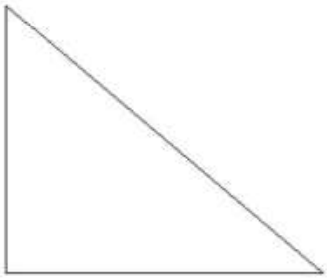
```
producer thread running
Waiting for return key.
Return key pressed
Resumed
```

(b) Create an Applet to draw a triangle on the basis of input given by user.

Ans.

```
import java.applet.*;
import java.awt.*;
public class Triangle extends Applet
{
public void paint(Graphics g1)
{
g1.drawLine(180,150,180,370);
g1.drawLine(180,150,440,370);
g1.drawLine(180,370,440,370);
}
}
<html>
<head>
</head>
<body>
<applet code = "Triangle.class" width = "640" height = "520"></applet>
</body>
</html>
```

Output:



Q7. (a) What is object serialization? Explain advantage of object serialization.

Ans. Object serialization: Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.

After a serialized object has been written into a file, it can be read from the file and deserialized that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.

Most impressive is that the entire process is JVM independent, meaning an object can be serialized on one platform and deserialized on an entirely different platform.

Classes ObjectInputStream and ObjectOutputStream are high-level streams that contain the methods for serializing and deserializing an object.

Advantage of object serialization: The advantages of serialization are:

It is easy to use and can be customized.

The serialized stream can be encrypted, authenticated and compressed, supporting the needs of secure Java computing.

Serialized classes can support coherent versioning and are flexible enough to allow gradual evolution of your application's object schema.

Serialization can also be used as a mechanism for exchanging objects between Java and C++ libraries, using third party vendor libraries (like RogueWave's Tools.h++) within C++.

There are simply too many critical technologies that rely upon serialization, including RMI, JavaBeans and EJB.

(b) What is need of layout manager? Explain different layouts available in java for GUI programming. Write code to set the layout of an applet.

Ans. Need of layout manager: The Layout Managers are used to arrange components in a particular manner. The Java Layout Managers facilitates us to control the positioning and size of the components in GUI forms. Layout Manager is an interface that is implemented by all the classes of layout managers.

Layouts available in java for GUI programming: There are 6 layout managers in Java

- **FlowLayout**: It arranges the components in a container like the words on a page. It fills the top line from left to right and top to bottom. The components are arranged in the order as they are added i.e. first components appears at top left, if the container is not wide enough to display all the components, it is wrapped around the line. Vertical and horizontal gap between components can be controlled. The components can be left, center or right aligned.
- **BorderLayout:** It arranges all the components along the edges or the middle of the container i.e. top, bottom, right and left edges of the area. The components added to the top or bottom gets its preferred height, but its width will be the width of the container and also the components added to the left or right gets its preferred width, but its height will be the remaining height of the container. The components added to the center gets neither its preferred height or width. It covers the remaining area of the container.
- **GridLayout**: It arranges all the components in a grid of equally sized cells, adding them from the left to right and top to bottom. Only one component can be placed in a cell and each region of the grid will have the same size. When the container is resized, all cells are automatically resized. The order of placing the components in a cell is determined as they were added.
- **GridBagLayout**: It is a powerful layout which arranges all the components in a grid of cells and maintains the aspect ration of the object whenever the container is resized. In this layout, cells may be different in size. It assigns a

consistent horizontal and vertical gap among components. It allows us to specify a default alignment for components within the columns or rows.

- **BoxLayout:** It arranges multiple components in either vertically or horizontally, but not both. The components are arranged from left to right or top to bottom. If the components are aligned horizontally, the height of all components will be the same and equal to the largest sized components. If the components are aligned vertically, the width of all components will be the same and equal to the largest width components.
- **CardLayout**: It arranges two or more components having the same size. The components are arranged in a deck, where all the cards of the same size and the only top card are visible at any time. The first component added in the container will be kept at the top of the deck. The default gap at the left, right, top and bottom edges are zero and the card components are displayed either horizontally or vertically.

Code to set the layout of an applet:

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

public class LayoutFlow extends JApplet {

  public void init() {
    //create a variable that references
    //the content pane of the applet
    Container appletContainer = getContentPane();

    // set the layout to FlowLayout
    appletContainer.setLayout( new FlowLayout() );

    // creates five buttons
    JButton button1 = new JButton("Button1");
    JButton button2 = new JButton(" Button2");
    JButton button3 = new JButton(" Button3");

    // add the buttons to the applet's content pane
    appletContainer.add(button1);
    appletContainer.add(button2);
    appletContainer.add(button3);
  }
}
```

Output:

Q8. (a) What is RMI? Explain its use.

Ans. RMI: The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.
The RMI provides remote communication between the applications using two objects stub and skeleton.

Use: RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a distributed object application.

(b) What is JDBC? Explain need of JDBC drivers.

Ans. JDBC: JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database.

Need of JDBC drivers: A JDBC driver uses the JDBC (Java Database Connectivity) API developed by Sun Microsystems, now part of Oracle, that provides a standard way to access data using the Java™ programming language. Using JDBC, an application can access a variety of databases and run on any platform with a Java Virtual Machine. It isn't necessary to write separate applications to access different database systems (Oracle and Salesforce, for example). Using JDBC allows you to write one application that can send SQL statements to different data sources. SQL is the standard language for accessing relational databases.

The JDBC API defines a set of Java interfaces that encapsulate major database functionality, such as running queries, processing results, and determining configuration information. Because JDBC applications are written in Java, applications work on any platform.

(c) What is Servlet? Explain use of GET and POST methods in Servlet.

Ans. Servlet: Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

Use of GET and POST methods in Servlet:

GET Method: The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? (question mark) symbol as follows –
http://www.test.com/hello?key1 = value1&key2 = value2

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.
This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using doGet() method.

POST Method: A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests sing doPost() method.