OBJECT ORIENTED PROGRAMMING USING JAVA

ASSIGNMENT

1.Explain various kinds of operators used in java ?

A) Java provides a variety of operators to perform different types of operations, such as arithmetic, logical, comparison, and assignment. Here's an overview of the key categories of operators in Java:

1. Arithmetic Operators

Used to perform mathematical operations.

- + : Addition
- - : Subtraction
- * : Multiplication
- / : Division
- % : Modulus (remainder of a division)

Example:

```
int a = 10, b = 5;
System.out.println(a + b);        // Output: 15
```

---

2. Relational (Comparison) Operators

Used to compare two values. The result is a boolean (true or false).

- == : Equal to
- !=   : Not equal to
- >   : Greater than
- <   : Less than
- >=   : Greater than or equal to
- <=   : Less than or equal to

Example:

```
int a = 10, b = 5;
System.out.println(a > b);     // Output: true
```

---

3. Logical Operators

Used to perform logical operations, typically in conditional statements.

- && : Logical AND
- || : Logical OR
- ! : Logical NOT

Example:

int a = 10, b = 5;

System.out.println(a > 5 && b < 10); // Output: true

---

4. Bitwise Operators

Operate at the bit level.

- & : Bitwise AND
- | : Bitwise OR
- ^ : Bitwise XOR
- ~ : Bitwise Complement
- << : Left shift
- >> : Right shift
- >>> : Unsigned right shift

Example:

int a = 5, b = 3;

System.out.println(a & b); // Output: 1 (binary AND)

---

5. Assignment Operators

Used to assign values to variables.

- = : Assign
- += : Add and assign
- -= : Subtract and assign
- *= : Multiply and assign
- /= : Divide and assign
- %= : Modulus and assign

Example:

int a = 10;

a += 5; // Equivalent to a = a + 5

System.out.println(a);   // Output: 15

---

6. Unary Operators

Operate on a single operand.

- +   : Unary plus
- -   : Unary minus
- ++   : Increment (pre/post)
- --   : Decrement (pre/post)
- !   : Logical complement

Example:

int a = 10;

System.out.println(++a);   // Pre-increment: Output 11

System.out.println(a--);   // Post-decrement: Output 11, then a becomes 10

---

7. Ternary Operator

A shorthand for if-else statements.

- condition ? value1 : value2

Example:

int a = 10, b = 20;

int max = (a > b) ? a : b;

System.out.println(max);   // Output: 20

---

8. Instance of Operator

Used to test whether an object is an instance of a specific class or subclass.

- object instanceof ClassName

Example:

String str = "Hello";

System.out.println(str instanceof String);   // Output: true

---

9. Type Casting Operator

Used to convert one data type into another.

- (datatype)

Example:

```
double d = 10.5;
int a = (int) d;      // Explicit casting
System.out.println(a);       // Output: 10
```

---

10. Special Operators

- **new** : Used to create objects.
- **.** : Used to access members of a class or object.
- **[]** : Used to access array elements.
- **::** : Method reference operator (introduced in Java 8).

Example:

```
String[] arr = {"A", "B", "C"};
System.out.println(arr[1]);          // Output: B
```

2. What is inheritance ? Explain it with suitable syntax and examples ?

A) Inheritance is one of the core concepts of Object-Oriented Programming (OOP). It allows a class (called the **child class** or **subclass**) to inherit the properties and methods of another class (called the **parent class** or **superclass**). This promotes code reuse, making programs easier to maintain and extend.

Example of Inheritance with Syntax:

```
// Parent Class
class Animal {
    String name;

    void eat() {
        System.out.println(name + " is eating.");
    }
}

// Child Class
class Dog extends Animal {
    void bark() {
        System.out.println(name + " is barking.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.name = "Buddy";    // Inheriting 'name' from Animal class
        dog.eat();          // Inheriting 'eat()' method from Animal class
        dog.bark();         // Calling its own 'bark()' method
    }
}
```

OUTPUT:

```
Buddy is eating.
Buddy is barking.
```

3. what are the uses of packages in java ? How to create and used it . Explain

A) In Java, a **package** is a way to organize classes and interfaces logically. It is essentially a directory or folder that groups related classes together. This helps in modularizing the code, preventing naming conflicts, and making it easier to manage and reuse code.

Uses of Packages in Java

1. **Organizing Classes:**
   Packages group related classes and interfaces together, making the code easier to maintain and understand.
2. **Avoiding Name Conflicts:**
   By using packages, classes with the same name can exist in different packages without conflict. For example:
   - java.util.Date

- java.sql.Date

3. **Access Control:**
   Packages allow you to define access levels (using public, protected, or default access modifiers), which helps in encapsulating and securing code.
4. **Code Reusability:**
   Once a package is created, it can be imported and reused in multiple projects.
5. **Easier Maintenance:**
   With a proper package structure, updating and managing code becomes simpler.
6. **Standardized Organization:**
   Packages follow a hierarchical structure (e.g., com.company.project.module) that mirrors the folder structure on disk.

How to Create and Use Packages in Java

Steps to Create a Package

1. **Create a Package**
   Use the package keyword at the top of the Java file.
2. **Save the File**
   Save the file in a directory that matches the package name.
3. **Compile the File**
   Use the javac command with the -d option to specify the base directory for the package structure.
4. **Import and Use the Package**
   Use the import statement to include the package in other classes.

4. Write a short notes on arrays

A) An array is a data structure in Java that allows you to store multiple values of the same type in a single container. Arrays are fixed in size and stored in contiguous memory locations.

Key Features of Arrays

1. **Fixed Size:** Once declared, the size of an array cannot be changed.
2. **Indexed Access:** Elements are accessed using an index starting from 0.
3. **Homogeneous Data:** All elements in an array must be of the same data type.
4. **Efficient Access:** Provides constant-time access to elements using their index.

5. What is Exception in java ? Explain exception handling

A) An **exception** in Java is an event that disrupts the normal flow of the program. It occurs during runtime and indicates that something unexpected has happened, such as invalid input, a file not found, or division by zero.

Exception handling in Java ensures that the program can recover gracefully from unexpected errors. Java provides the following key constructs:

1. try Block

The code that might throw an exception is placed inside the try block.

2. catch Block

The code to handle the exception is placed inside the catch block. It catches specific exceptions.

3. finally Block

The finally block contains code that always executes, regardless of whether an exception occurred or not (e.g., closing resources).

4. throw Keyword

Used to explicitly throw an exception.

5. throws Keyword

Used in a method signature to declare exceptions that a method can throw.

6.Expain about collection classes with example

A) Java collection class is used exclusively with static methods that operate on or return collections. It inherits Object class.

The important points about Java Collections class are:

- Java Collection class supports the **polymorphic algorithms** that operate on collections.
- Java Collection class throws a **NullPointerException** if the collections or class objects provided to them are null.

```
import java.util.*;
public class CollectionsExample {
    public static void main(String a[]){
        List<String> list = new ArrayList<String>();
        list.add("C");
        list.add("Core Java");
        list.add("Advance Java");
        System.out.println("Initial collection value:"+list);
        Collections.addAll(list, "Servlet","JSP");
        System.out.println("After adding elements collection value:"+list);
        String[] strArr = {"C#", ".Net"};
o          Collections.addAll(list, strArr);
o          System.out.println("After adding array collection value:"+list);
o      }
o   }
```

OUTPUT:

Initial collection value:[C, Core Java, Advance Java]

After adding elements collection value:[C, Core Java, Advance Java, Servlet, JSP]

After adding array collection value:[C, Core Java, Advance Java, Servlet, JSP, C#, .Net]

7.Explain about AWT classes

A) **AWT (Abstract Window Toolkit)** is a part of Java's java.awt package and provides a set of APIs for creating graphical user interface (GUI) components like windows, buttons, text fields, etc. It is one of the oldest GUI toolkits in Java and is platform-dependent because it uses the native GUI of the operating system.

8.Explain about date class and timer class methods in java

A) **Date Class in Java**

- Part of java.util.
- Represents date and time.
- Key Methods:
    - getTime(): Returns milliseconds since epoch.
    - before(Date), after(Date): Compare dates.
    - toString(): Converts date to string.

Timer Class in Java

- Part of java.util.
- Schedules tasks for one-time or repeated execution.
- Key Methods:
    - schedule(TimerTask, delay): Executes after delay.
    - scheduleAtFixedRate(TimerTask, delay, period): Repeats task.

9.How to do event handling in java . Explain

A) Event handling in Java allows programs to respond to user interactions (like clicks, key presses, etc.) with GUI components such as buttons, text fields, etc. It follows the **event-driven programming** model.

10.Write the usage of layout managers and menus in java

A) Usage of Layout Managers in Java

Layout managers are used to control the arrangement of components in a container (like Frame or Panel). They help in automatically adjusting the size and position of components based on the layout rules, ensuring a consistent and responsive UI design. Common layouts include:

- **FlowLayout**: Used for simple layouts where components flow from left to right.
- **BorderLayout**: Ideal for applications with regions like top, bottom, left, right, and center.
- **GridLayout**: Used for arranging components in a fixed grid of rows and columns.
- **CardLayout**: Useful for tabbed or card-based layouts where only one component is visible at a time.
- **GridBagLayout**: Suitable for complex layouts requiring flexibility in row and column sizes.

Usage of Menus in Java

Menus are used to provide users with options like File, Edit, and Help. They enhance user interaction and navigation in applications.

- **MenuBar**: Contains the menus.
- **Menu**: Groups related items (e.g., File, Edit).
- **MenuItem**: Represents an individual item in the menu (e.g., Open, Save).

Menus are added to the frame's MenuBar, and actions are handled using listeners (like ActionListener) to perform operations when items are selected.

11.Write about Swing and its Architecture

A) Swing is a part of the javax.swing package used to create graphical user interfaces (GUIs) in Java. It provides a rich set of lightweight, platform-independent GUI components like buttons, labels, text fields, and tables, offering greater flexibility and customization compared to AWT.

Swing Architecture

Swing's architecture is built around the **Model-View-Controller (MVC)** pattern:

1. **Model**: Represents the data or state of the component.
2. **View**: Handles the visual display of the component.
3. **Controller**: Manages user input and updates the view.

12.Explain about Java Data Output Stream Class

A) Java Data Output Stream Class :

The DataOutputStream class in Java, part of the java.io package, allows writing primitive data types (like int, float, double, boolean, etc.) to an output stream in a machine-independent, binary format. It is commonly used for saving data in a binary format for files or network communication.

Key Features:

- Writes primitive data types in binary format.
- Ensures portability across platforms.
- Provides methods like writeInt(), writeFloat(), writeDouble(), and writeUTF() for various data types.

13.Explain about usage of Soket Programming in Java

A) **Usage of Socket Programming in Java**

Socket programming in Java is used to enable communication between computers over a network. It is commonly used in the following scenarios:

1. **Client-Server Applications**: Java socket programming is used to build client-server systems where multiple clients communicate with a centralized server, such as:
   - Chat applications
   - Web servers (HTTP)
   - Online gaming
   - File transfer applications
2. **Real-time Communication**: It allows continuous and real-time data exchange between clients and servers, such as:
   - Video conferencing
   - Live streaming
   - Real-time collaborative applications
3. **Networking Services**: Socket programming enables services like:
   - Database access over a network
   - Remote procedure calls (RPC)

   - Distributed systems communication

4. **IoT Applications:** Sockets allow devices in an Internet of Things (IoT) network to communicate, share data, and interact in real-time.

14.What is Stream ? Write about Various I/O classes

A) In Java, **streams** are used to read and write data to and from various sources like files, memory, or network connections. Streams provide a consistent way to handle input and output (I/O) operations.

Key I/O Classes:

- **FileInputStream/FileOutputStream:** For reading/writing binary data to/from files.
- **FileReader/FileWriter:** For reading/writing text data to/from files.
- **BufferedReader/BufferedWriter**: For efficient reading and writing of text.
- **DataInputStream/DataOutputStream**: For reading/writing primitive data types.
- **PrintStream**: For printing formatted text, often to files or the console.
- **ObjectInputStream/ObjectOutputStream**: For object serialization and deserialization.

15.Explain about Serialization and Deserialization in java

A) **Serialization and Deserialization in Java**

Serialization is the process of converting an object's state into a byte stream so it can be easily saved to a file, sent over a network, or stored in a database.

Deserialization is the reverse process of converting the byte stream back into an object, allowing it to be restored to its original state.

Key Points:

- **Serializable Interface:** A class must implement the java.io.Serializable interface for its objects to be serialized.
- **ObjectOutputStream:** Used for serializing an object.
- **ObjectInputStream:** Used for deserializing an object.