WEEK - 03 ASSIGNMENT

Short Answer Type:

1. What is the difference between a compiler and Interpreter?

   The processor processes the entire program at once and gets executed faster after compilation. The Interpreter processes line by line and has slower execution as it interprets during runtime.

2. What are the features of object oriented programming languages?

   The features of OOP are:

   1. Encapsulation: Wrapping of data members and member functions.
   2. Inheritance: Allows a class to inherit properties and behaviors from parent classes.
   3. Polymorphism: Enables objects to take different forms depending on the functionality.
   4. Abstraction: Hides complex implementation details and shows only necessary features.
   5. Class: Blueprint for creating objects
   6. Object: Instance of a class that has states and behaviors
   7. Method Overloading: Multiple methods with same name but different parameters
   8. Method Overriding: Subclass providing specific implementation for method defined in parent class.

3. What is bytecode?

   Intermediate code format generated when Java source code is compiled by the compilers.

4. What is JVM?

   A Java Virtual Machine (JVM) is a virtual machine that executes Java bytecode, a platform-independent intermediate code. It provides a runtime environment for Java applications to run on various operating systems and hardware platforms.

5. Write different types of operators in java?

   - Arithmetic Operators: +, -, *, /, % (modulo)
   - Relational Operators: ==, !=, >, <, >=, <=
   - Logical Operators: && (AND), || (OR), ! (NOT)
   - Assignment Operators: =, +=, -=, *=, /=, %=, etc.
   - Increment/Decrement: ++ (increment), -- (decrement)
   - Bitwise Operators: & (AND), | (OR), ^ (XOR), ~ (complement), <<, >>, >>> (shift)
   - Conditional (Ternary): ? : (condition ? expression1 : expression2)

6. What is java token and different data types?

Java tokens are the basic building blocks of a Java program. They are the smallest unit of a program, and they include keywords, identifiers, operators, literals, separators, and comments.

1. byte (8-bit, -128 to 127)
2. short (16-bit, -32,768 to 32,767)
3. int (32-bit, -2^31 to 2^31-1)
4. long (64-bit, -2^63 to 2^63-1)
5. float (32-bit, single-precision)
6. double (64-bit, double-precision)
7. char (16-bit, Unicode character)
8. boolean (true/false)

7. What are different types of variables in java?

- Local Variables: Declared inside methods, constructors, or blocks; must be initialized before use
- Instance Variables: Declared in a class but outside any method; created when object is instantiated
- Static/Class Variables: Declared with static keyword; shared among all instances of the class
- Reference Variables: Hold references to objects rather than the objects themselves
- Final Variables: Cannot be modified after initialization (constants)

8. What are different control structures or statements in Java?

- Decision-Making Statements: if statement, if-else statement, if-else-if ladder, switch statement, conditional operator (? :)
- Loop Statements: for loop, enhanced for loop (for-each), while loop, do-while loop
- Jump Statements: break, continue, return

9. What are jumping statements in Java?

- break: Terminates the innermost loop or switch statement

  for(int i=0; i<10; i++) {

  if(i == 5) break; } *// exits the loop when i equals 5*

- continue: Skips the current iteration and proceeds to the next iteration

  for(int i=0; i<10; i++) {

  if(i == 5) continue; } *// skips iteration when i equals 5*

- return: Exits from a method with or without a value

  int sum(int a, int b) {

  return a + b; } *// returns the sum and exits the method*

10. Write difference between class and object?

    Class is the blueprint of an object. It is declared once and contains abstract definitions of attributes and behaviors.

    Object is an instance of a class. Multiple objects can be created from a class and occupies memory when instantiated.

12. What is a Constructor ? and Types of it. Give an example?

    Constructor: Special method used to initialize objects when they are created.
    Types of Constructors:

    1. Default Constructor: No parameters, provides default values

```
class Student {
    // Default constructor
    Student() {
        System.out.println("Student object created");
    }
}
```

    2. Parameterized Constructor: Accepts parameters to initialize fields

```
class Student {
    String name;
    int id;
    // Parameterized constructor
    Student(String name, int id) {
        this.name = name;
        this.id = id;
    }
}
```

    3. Copy Constructor: Creates object using another object of the same class

```
class Student {
    String name;
    int id;
    // Regular constructor
    Student(String name, int id) {
        this.name = name;
```

```
        this.id = id;
    }
    // Copy constructor
    Student(Student s) {
        this.name = s.name;
        this.id = s.id;
    }
}
```

13. What are Access Modifiers? Give an example?

    Access Modifiers: Keywords that determine the scope and visibility of classes, methods, and variables.

    1. public: Accessible from any class

       ```
       public class PublicClass {
           public int publicVar = 10;
           public void publicMethod() { }
       }
       ```

    2. protected: Accessible within the same package and subclasses

       ```
       class Parent {
           protected int protectedVar = 20;
           protected void protectedMethod() { }
       }
       ```

    3. default (no modifier): Accessible only within the same package

       ```
       class DefaultClass {
           int defaultVar = 30;
           void defaultMethod() { }
       }
       ```

    4. private: Accessible only within the declared class

       ```
       class PrivateExample {
           private int privateVar = 40;
           private void privateMethod() { }
       }
       ```

14. What is a wrapper class?

- Wrapper Classes: Provide way to use primitive data types as objects
- Convert primitive types to reference types (boxing) and vice versa (unboxing)
- Enable primitive types to be used where objects are required
- Include methods for parsing and conversion

15. What is an inner class?

Inner Class: A class defined within another class. Has access to the outer class's members, including private members. Four types of inner classes in Java:

Long Answer Type Questions

1. Write the advantages and all features of object oriented programming? Explain it?

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects" that contain members and functions. It offers numerous advantages and features:

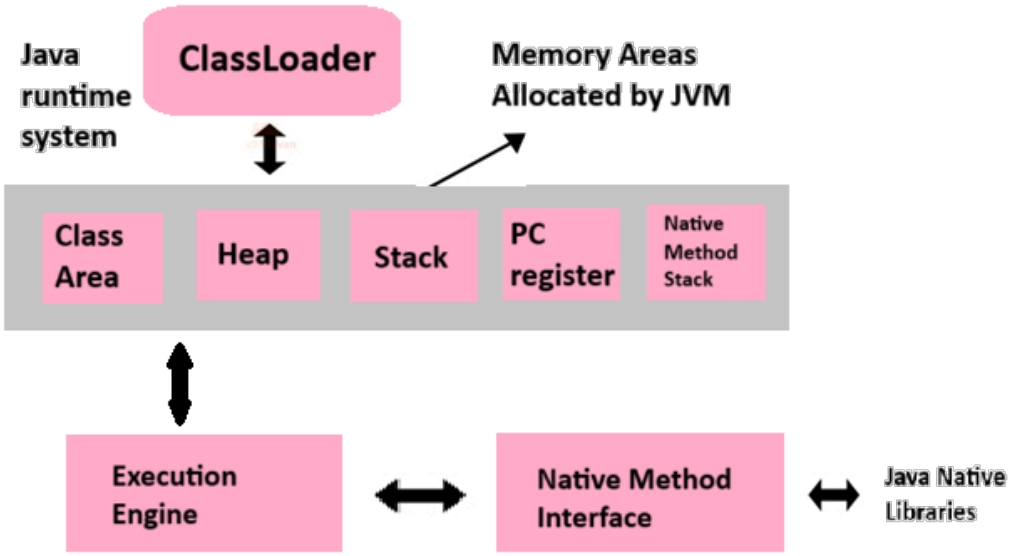
Features of OOP:

- Encapsulation: Wraps data and methods into a single unit (class), hiding internal states and requiring all interactions to occur through well-defined interfaces
- Inheritance: Allows a class to inherit properties and behaviors from a parent class, promoting code reuse and establishing a hierarchical relationship
- Polymorphism: Enables objects to take multiple forms depending on context, allowing methods to do different things based on the object using them or parameters passed
- Abstraction: Hides complex implementation details and shows only necessary features to the user, reducing complexity
- Class and Object: Class serves as a blueprint while objects are instances created from that blueprint.

Advantages of OOP:

- Increased Productivity: We can create programs from pre-written, interconnected modules rather than starting from scratch, saving time and increasing productivity
- Modularity: Breaks software into manageable, discrete problems, allowing for division of labor in development
- Extensibility: Allows adding new characteristics and actions to objects without major program modifications
- Reusability: Objects can be utilized in multiple applications, reducing redundant code
- Code Organization: Makes code more maintainable and logical by organizing it into objects that model real-world entities
- Security: Through encapsulation, data can be hidden and protected from accidental manipulation

2. Draw complete architecture of JVM. Describe the use of each component in JVM?



The JVM is an abstract computing machine that enables Java programs to be platform-independent by converting bytecode into machine-specific instructions. Below is the complete architecture of JVM along with the description of each component:

1. Class Loader Subsystem
Responsible for loading, linking, and initializing Java classes.
Loading: Reads .class files and loads them into the method area.

Linking:
Verification: Checks bytecode validity.
Preparation: Allocates memory for static variables.
Resolution: Converts symbolic references to direct references.
Initialization: Executes static initializers (static blocks).

Types of Class Loaders:
Bootstrap ClassLoader: Loads core Java classes (rt.jar).
Extension ClassLoader: Loads classes from jre/lib/ext.
Application ClassLoader: Loads user-defined classes from the classpath.

2. Runtime Data Areas (Memory Areas)
Stores data during program execution.

a) Method Area
Stores class metadata, static variables, and method bytecode.
Shared among all threads.

b) Heap Area
Stores all objects and arrays (created using new).
Garbage Collection works here.

Divided into:
Young Generation (Eden, S0, S1) – Short-lived objects.
Old Generation (Tenured Space) – Long-lived objects.
Permanent Generation (MetaSpace in Java 8+) – Class metadata.

c) Stack Area (Per Thread)
Stores method calls, local variables, and partial results.
Each thread has its own stack.
Divided into stack frames (one per method call).

d) PC Register (Program Counter)
Holds the address of the currently executing instruction.
Each thread has its own PC register.

e) Native Method Stack
Stores native method (C/C++) calls.

3. Execution Engine
Executes bytecode.

a) Interpreter
Reads and executes bytecode line by line.
Slower execution.

b) JIT Compiler (Just-In-Time Compiler)
Compiles frequently used bytecode into native machine code for faster execution.
HotSpot Detection: Identifies "hot" code for optimization.

c) Garbage Collector (GC)
Automatically reclaims unused memory (heap).
Runs in the background (Daemon Thread).

4. Native Method Interface (JNI)
Allows Java to interact with native libraries (C/C++).
Used in java.lang, java.io, etc.

5. Native Method Libraries
Contains native libraries (.dll, .so) required by JVM.

3. Describe different data types and operators in java with all examples?

Primitive Data Types:

```
byte x = 99;
System.out.println(x);  // Output: 99

short x = 999;
System.out.println(x);  // Output: 999

int x = 99999;
System.out.println(x);  // Output: 99999

long x = 99999999999 L;
System.out.println(x);  // Output: 99999999999

float x = 99.99f;
System.out.println(x);  // Output: 99.99

double x = 99.99d;
System.out.println(x);  // Output: 99.99

char division = 'A';
System.out.println(division);  // Output: A

boolean isAvailable = true;
System.out.println(isAvailable);
```

Arithmetic Operators:

```
int a = 10;
int b = 5;
System.out.println("Addition: " + (a + b));       // Output: 15
System.out.println("Subtraction: " + (a - b));    // Output: 5
System.out.println("Multiplication: " + (a * b)); // Output: 50
System.out.println("Division: " + (a / b));       // Output: 2
System.out.println("Modulus: " + (a % b));        // Output: 0
```

Relational Operators

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

```
System.out.println(a >= b); // Output: true
System.out.println(a <= b); // Output: false
```

Logical Operators

```
boolean x = true;
boolean y = false;
System.out.println(x && y); // Output: false (AND)
System.out.println(x || y); // Output: true (OR)
System.out.println(!x);     // Output: false (NOT)
```

Assignment Operators

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

a -= 3;  // a = a - 3
System.out.println(a);  // Output: 12

a *= 2;  // a = a * 2
System.out.println(a);  // Output: 24

a /= 4;  // a = a / 4
System.out.println(a);  // Output: 6

a %= 4;  // a = a % 4
System.out.println(a);  // Output: 2
```

Increment/Decrement Operators:

```
int a = 5;
System.out.println(a++);  // Output: 5 (post-increment)
System.out.println(a);    // Output: 6
System.out.println(++a);  // Output: 7 (pre-increment)
System.out.println(a--);  // Output: 7 (post-decrement)
System.out.println(a);    // Output: 6
System.out.println(--a);  // Output: 5 (pre-decrement)
```

Bitwise Operators:

```
int a = 5;  // 101 in binary
int b = 3;  // 011 in binary
System.out.println(a & b);   // Output: 1 (001 in binary)
System.out.println(a | b);   // Output: 7 (111 in binary)
System.out.println(a ^ b);   // Output: 6 (110 in binary)
```

```
System.out.println(~a);      // Output: -6 (complement)
System.out.println(a << 1);  // Output: 10 (left shift)
System.out.println(a >> 1);  // Output: 2 (right shift)
```

Ternary Operator

```
int a = 10;
int b = 5;
int min = (a < b) ? a : b;
System.out.println(min);  // Output: 5
```

4. Describe the use of Access Modifiers in Java? Draw a table showing the scope of accessibility?

Access modifiers in Java control the visibility and accessibility of classes, methods, and variables. They determine which parts of a program can access specific elements of your code

Types of Access Modifiers:

Modifier	Class	Package	Subclass	World
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
default (no modifier)	Yes	Yes	No	No
private	Yes	No	No	No

Explanation:

1. public: The member is accessible from any class anywhere
2. protected: The member is accessible within its own package and in subclasses (even if they're in different packages)

3. default (no keyword): The member is accessible only within its own package
4. private: The member is accessible only within its own class

5. Write programs to show the use of access modifiers?

1. Public Access Modifier

```
class PublicExample {
    public int publicVar = 10;

    public void publicMethod() {
        System.out.println("Public method");
    }
}

public class Main {
    public static void main(String[] args) {
        PublicExample obj = new PublicExample();
        System.out.println(obj.publicVar);
        obj.publicMethod();
    }
}
```

2. Private Access Modifier

```
class PrivateExample {
    private int privateVar = 20;

    private void privateMethod() {
        System.out.println("Private method");
    }

    public void accessPrivate() {
        System.out.println(privateVar);
        privateMethod();
    }
}

public class Main {
    public static void main(String[] args) {
        PrivateExample obj = new PrivateExample();
        obj.accessPrivate();
    }
}
```

3. Protected Access Modifier

```
class Parent {
    protected int protectedVar = 30;
```

```
    protected void protectedMethod() {
        System.out.println("Protected method");
    }
}

class Child extends Parent {
    void accessProtected() {
        System.out.println(protectedVar);
        protectedMethod();
    }
}

public class Main {
    public static void main(String[] args) {
        Child obj = new Child();
        obj.accessProtected();
    }
}
```

4. Default Access Modifier

```
class DefaultExample {
    int defaultVar = 40;

    void defaultMethod() {
        System.out.println("Default method");
    }
}

public class Main {
    public static void main(String[] args) {
        DefaultExample obj = new DefaultExample();
        System.out.println(obj.defaultVar);
        obj.defaultMethod();
    }
}
```

6. Write a program to show the use of constructors calling each other?

```
public class ConstructorChainingExample {
    private String name;
    private int age;
    private String department;
    public ConstructorChainingExample() {
```

```
    this("Unknown");
    System.out.println("Default constructor called");
}
public ConstructorChainingExample(String name) {
    this(name, 0);
    System.out.println("Constructor with name called");
}
public ConstructorChainingExample(String name, int age) {
    this(name, age, "Undefined");
    System.out.println("Constructor with name and age called");
}
public ConstructorChainingExample(String name, int age, String department) {
    this.name = name;
    this.age = age;
    this.department = department;
    System.out.println("Full parameter constructor called");
}
public void displayInfo() {
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Department: " + department);
}
public static void main(String[] args) {
    System.out.println("Creating object with default constructor:");
    ConstructorChainingExample obj1 = new ConstructorChainingExample();
    obj1.displayInfo();

    System.out.println("\nCreating object with name parameter:");
    ConstructorChainingExample obj2 = new ConstructorChainingExample("Alice");
    obj2.displayInfo();
```

```
        System.out.println("\nCreating object with name and age parameters:");
        ConstructorChainingExample obj3 = new ConstructorChainingExample("Bob", 25);
        obj3.displayInfo();

        System.out.println("\nCreating object with all parameters:");
        ConstructorChainingExample obj4 = new ConstructorChainingExample("Charlie",
    30, "Engineering");
        obj4.displayInfo();
    }
}
```

7. Write a program to show the use of constructor overloading?

```
class Rectangle {
    int length;
    int width;
    Rectangle() {
        length = 1;
        width = 1;
    }
    Rectangle(int side) {
        length = side;
        width = side;
    }
    Rectangle(int l, int w) {
        length = l;
        width = w;
    }
    void display() {
        System.out.println("Length: " + length + ", Width: " + width);
    }
}
public class Main {
```

```
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5);
        Rectangle r3 = new Rectangle(3, 4);

        r1.display();
        r2.display();
        r3.display();
    }
}
```

8. Write short notes on:

a. Inner class

An inner class in Java is a class defined within another class11. They enable logical grouping of classes that are used in only one place, increased encapsulation, and more readable and maintainable code.

```
class OuterClass {
    int x = 10;
    class InnerClass {
        int y = 5;
        void innerMethod() {
            System.out.println("Inner method can access outer x: " + x);
        }
    }
    void outerMethod() {
        InnerClass inner = new InnerClass();
        System.out.println("Inner y accessed from outer method: " + inner.y);
        inner.innerMethod();
    }
}

public class InnerClassDemo {
    public static void main(String[] args) {
```

```
        OuterClass outer = new OuterClass();
        OuterClass.InnerClass inner = outer.new InnerClass();
        System.out.println("Accessing inner class y: " + inner.y);
        inner.innerMethod();
        outer.outerMethod();
    }
}
```

b. Member Inner class

A member inner class is defined at the member level of the outer class. It has the following characteristics:

- Can access all members of the outer class including private members
- Can be declared with any access modifier (public, private, protected, default)
- Cannot contain static members if the inner class itself is not static
- Can extend different classes or implement interfaces

```
class OuterClass {
    private int outerField = 10;
    public class MemberInnerClass {
        private int innerField = 20;
        public void display() {
            System.out.println("Outer field: " + outerField);
            System.out.println("Inner field: " + innerField);
            outerMethod();
        }
    }
    private void outerMethod() {
        System.out.println("Outer method called");
    }
    public void createInner() {
        MemberInnerClass inner = new MemberInnerClass();
        inner.display();
    }
}
```

```
public class MemberInnerClassDemo {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        OuterClass.MemberInnerClass inner = outer.new MemberInnerClass();
        inner.display();
        System.out.println("\nUsing createInner method:");
        outer.createInner();
    }
}
```

c. Static Inner class

A static inner class (also called static nested class) is a static member of the outer class and behaves like a top-level class, with the following characteristics:

- Cannot access non-static members of the outer class directly
- Can be accessed without an instance of the outer class
- Can contain both static and non-static members
- Can access static members of the outer class directly

java

```
class OuterClass {
    private static int staticOuterField = 10;
    private int instanceOuterField = 20;
    static class StaticInnerClass {
        private int innerField = 30;
        private static int staticInnerField = 40;
        public void display() {
            System.out.println("Static outer field: " + staticOuterField);
            OuterClass outer = new OuterClass();
            System.out.println("Instance outer field: " + outer.instanceOuterField);
            System.out.println("Inner field: " + innerField);
            System.out.println("Static inner field: " + staticInnerField);
        }
        public static void staticDisplay() {
            System.out.println("Static method in static inner class");
```

```
            System.out.println("Static outer field: " + staticOuterField);
            System.out.println("Static inner field: " + staticInnerField);
        }
    }
}
public class StaticInnerClassDemo {
    public static void main(String[] args) {
        OuterClass.StaticInnerClass inner = new OuterClass.StaticInnerClass();
        inner.display();
        System.out.println("\nCalling static method:");
        OuterClass.StaticInnerClass.staticDisplay();
    }
}
```

d. Local Inner class

A local inner class is defined within a method or block, with the following characteristics:

- Can access all members of the enclosing class
- Can access local variables of the enclosing method if they are final or effectively final
- Cannot be declared with access modifiers
- Cannot have static members
- Is not visible outside the method or block

```
class OuterClass {
    private int outerField = 10;
    public void method(final int param) {
        final int localVar = 20;
        int effectivelyFinalVar = 30;
        class LocalInnerClass {
            private int innerField = 40;
            public void display() {
                System.out.println("Outer field: " + outerField);
                System.out.println("Method parameter: " + param);
                System.out.println("Local variable: " + localVar);
                System.out.println("Effectively final variable: " + effectivelyFinalVar);
```

```
            System.out.println("Inner field: " + innerField);
        }
    }
    LocalInnerClass inner = new LocalInnerClass();
    inner.display();
  }
}
public class LocalInnerClassDemo {
  public static void main(String[] args) {
    OuterClass outer = new OuterClass();
    outer.method(50);
  }
}
```

9. Explain in detail about Static modifiers

The static keyword in Java is used to create class-level members (variables and methods) that belong to the class itself rather than to instances of the class.

Static Variables:

- Also known as class variables
- Shared among all instances of a class
- Initialized only once, at the start of execution
- Accessed using the class name: ClassName.variableName
- Stored in the class area of memory, not in objects

Static Methods:

- Belong to the class rather than objects
- Can access static variables directly
- Cannot access instance variables or call non-static methods directly
- Can be called using the class name: ClassName.methodName()
- Cannot use this or super keywords

Static Block:

- Executed when the class is loaded into memory
- Used for static initializations
- Executed before the main method
- Can be multiple static blocks in a class (executed in order)

Static Class:

- Only inner classes can be static
- Does not need an instance of the outer class to be created
- Cannot access non-static members of the outer class directly

10. Explain Formatters and Regex.

1. Formatters

Formatters are used to format data (numbers, dates, strings / MessageFormat) in a specific pattern for display or storage. Formate uses format() method to display the data in a more structured format.

Types of Formatters in Java:

a) String.format(): Formats strings using format specifiers (%s, %d, %f, etc.).

```
String name = "Alice";
int age = 25;
double salary = 50000.50;
String formatted = String.format("Name: %s, Age: %d, Salary: %.2f", name, age, salary);
System.out.println(formatted);  // Output: Name: Alice, Age: 25, Salary: 50000.50
```

b) DecimalFormat (Number Formatting): Formats numbers with custom patterns.

```
import java.text.DecimalFormat;
double num = 12345.6789;
DecimalFormat df = new DecimalFormat("#,##0.00");
System.out.println(df.format(num));  // Output: 12,345.68
```

c) SimpleDateFormat (Date Formatting): Formats dates into readable strings.

```
import java.text.SimpleDateFormat;
import java.util.Date;
Date now = new Date();
SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
System.out.println(sdf.format(now));  // Output: 28-04-2025 15:30:45
```

2. Regular Expressions (Regex) : Regex is a pattern-matching syntax used for searching, validating, and manipulating text.

a) Pattern & Matcher: Pattern is the compiled regex and matcher are engines that interpret patterns to locate matches in character sequence.

```
import java.util.regex.*;

String text = "Hello, my email is user@example.com";

String regex = "\\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}\\b";

Pattern pattern = Pattern.compile(regex);

Matcher matcher = pattern.matcher(text);


if (matcher.find()) {

   System.out.println("Email found: " + matcher.group());  // Output: Email found: user@example.com

}
```

b) String.matches(): Checks if a string matches a regex pattern.

```
String phone = "123-456-7890";

boolean isValid = phone.matches("\\d{3}-\\d{3}-\\d{4}");

System.out.println(isValid);  // Output: true
```

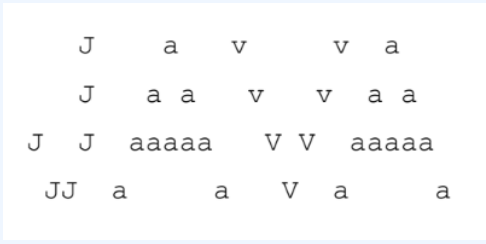
c) String.split(): Splits a string based on a regex pattern.

```
String data = "Java,Python,C++,JavaScript";

String[] languages = data.split(",");

for (String lang : languages) {

   System.out.println(lang);

}

// Output:

// Java

// Python

// C++

// JavaScript
```

Programming Exercises

1. Write a Java program to display the following pattern.

Sample Pattern :

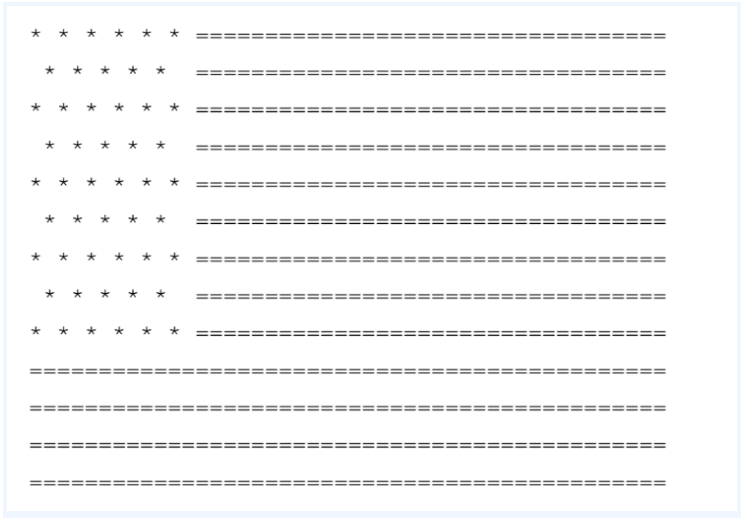



```
public class Program1 {
   static void javaPattren() {
      for (int i = 0; i < 4; i++) {
         for (int j = 0; j < 4; j++) {
            if ((j == 3 && i != 3) || (i == 2 && j == 0) ||
               (i == 3 && (j == 1 || j == 2))) {
               System.out.print("J");
            } else {
               System.out.print(" ");
            }
         }
         for (int j = 0; j < 3 - i; j++) {
            System.out.print(" ");
         }
         for (int k = 0; k < 2 * i + 1; k++) {
            if ((k == (2 * i + 1) / 2 && i == 0) || i == 2 ||
               (i == 1 && k == (2 * i + 1) / 2 + 1) ||
               (i == 1 && k == 0) || (i == 3 && (k == 0 || k == 2 * i))) {
               System.out.print("a");
            } else {
               System.out.print(" ");
            }
         }
```

```
            for (int j = 0; j < 3 - i; j++) {
                System.out.print(" ");
            }
            for (int j = 0; j < 2 * 4 - 1; j++) {
                if (j == i || j == (2 * 4 - 2 - i)) {
                    System.out.print("V");
                } else {
                    System.out.print(" ");
                }
            }
            for (int j = 0; j < 3 - i; j++) {
                System.out.print(" ");
            }
            for (int k = 0; k < 2 * i + 1; k++) {
                if ((k == (2 * i + 1) / 2 && i == 0) || i == 2 ||
                    (i == 1 && k == (2 * i + 1) / 2 + 1) ||
                    (i == 1 && k == 0) || (i == 3 && (k == 0 || k == 2 * i))) {
                    System.out.print("a");
                } else {
                    System.out.print(" ");
                }
            }
            for (int j = 0; j < 3 - i; j++) {
                System.out.print(" ");
            }
            System.out.println();
        }
    }

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

2. Write a Java program to print an American flag on the screen.

Expected Output




```
public class Program2 {
    static void pattern(int n) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n * 2; j++) {
                if (i % 2 == 0) {
                    if (j % 2 == 0) {
                        System.out.print("*");
                    } else {
                        System.out.print(" ");
                    }
                } else {
                    if (j % 2 == 0) {
                        System.out.print(" ");
                    } else {
                        System.out.print("*");
                    }
                }
            }
            for (int k = 0; k < n * 2; k++) {
                System.out.print("=");
            }
```

```
            System.out.println();
        }
        for (int i = 0; i < n - 3; i++) {
            for (int j = 0; j < n * 4; j++) {
                System.out.print("=");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int n = 6;
        pattern(n);
    }
}
```

3. Write a Java program to convert an integer number to

i. A binary number.

ii. Hexadecimal

iii. Octal

iv. Vice a versa

```
public class Conversion {
    public static void main(String[] args) {
        int number = 255; // Example number to convert

        // Convert to different bases
        System.out.println("Original number: " + number);

        // i. Convert to binary
        String binary = Integer.toBinaryString(number);
```

```
        System.out.println("Binary: " + binary);

        // ii. Convert to hexadecimal
        String hex = Integer.toHexString(number);
        System.out.println("Hexadecimal: " + hex);

        // iii. Convert to octal
        String octal = Integer.toOctalString(number);
        System.out.println("Octal: " + octal);

        // iv. Vice versa conversions
        System.out.println("\nReverse conversions:");

        // Binary back to integer
        int fromBinary = Integer.parseInt(binary, 2);
        System.out.println("Binary " + binary + " => " + fromBinary);

        // Hexadecimal back to integer
        int fromHex = Integer.parseInt(hex, 16);
        System.out.println("Hexadecimal " + hex + " => " + fromHex);

        // Octal back to integer
        int fromOctal = Integer.parseInt(octal, 8);
        System.out.println("Octal " + octal + " => " + fromOctal);
    }
}
```

```
Original number: 255
Binary: 11111111
Hexadecimal: ff
Octal: 377
Reverse conversions:
Binary 11111111 ? 255
Hexadecimal ff ? 255
Octal 377 ? 255
```

4. Write a Java program to count letters, spaces, numbers and other characters in an input string using Regex.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class CharacterCounter {
    public static void main(String[] args) {

        String inputString = "!Java1 Java2 Java3!";
        System.out.println("Entered string is : \"!Java1 Java2 Java3!\" ");

        int letters = countMatches(inputString, "[a-zA-Z]");
        int spaces = countMatches(inputString, "\\s");
        int numbers = countMatches(inputString, "\\d");
        int otherChars = inputString.length() - (letters + spaces + numbers);

        System.out.println("\nCharacter Counts");
        System.out.println("Letters: " + letters);
        System.out.println("Spaces: " + spaces);
        System.out.println("Numbers: " + numbers);
        System.out.println("Other Characters: " + otherChars);
    }

    private static int countMatches(String input, String regex) {
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(input);
        int count = 0;
        while (matcher.find()) {
            count++;
        }
        return count;
    }
}
```

5. Write a Java method to find the smallest number among three numbers.

```
public class FindSmallest {
    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 15;
        int num3 = 9;

        int smallest = findSmallest(num1, num2, num3);
        System.out.println("The smallest number is: " + smallest);
    }
    public static int findSmallest(int a, int b, int c) {
        return Math.min(a, Math.min(b, c));
    }
}
```

6. Write a Java method to compute the future investment value at a given interest rate for a specified number of years. Sample data (Monthly compounded) and Output:

Input the investment amount: 1000

Input the rate of interest: 10

Expected Output:

Years FutureValue

1 1104.71

2 1220.39

3 1348.18

4 1489.35

5 1645.31

```
public class FutureInvestment {
    public static void main(String[] args) {
        double investmentAmount = 1000;
        System.out.print("Investment amount: " + investmentAmount);
        double annualInterestRate = 10;
        System.out.print("Rate of interest: " + annualInterestRate);
```

```
        System.out.println("\nYears FutureValue");
        for (int year = 1; year <= 5; year++) {
            double futureValue = calculateFutureValue(investmentAmount, annualInterestRate, year);
            System.out.printf("%d %.2f%n", year, futureValue);
        }
    }

    public static double calculateFutureValue(double investmentAmount, double
annualInterestRate, int years) {
        double monthlyInterestRate = annualInterestRate / 100 / 12;
        int numberOfMonths = years * 12;
        double futureValue = investmentAmount * Math.pow(1 + monthlyInterestRate,
numberOfMonths);
        return futureValue;
    }
}
```

7. Write a Java method to check whether a string is a valid password.

Password rules: [Using Regex]

A password must have at least ten characters.

A password consists of only letters and digits.

A password must contain at least two digits.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ValidPassword {
    static boolean isValidPassword(String password) {
        String regex = "^(?=.*[0-9]{2,})(?=.*[a-zA-Z])[a-zA-Z0-9]{10,}$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(password);
        return matcher.matches();
    }
```

```
    public static void main(String[] args) {
        String password = "buzz22601MAAN**!";
        if(isValidPassword(password)) {
        System.out.println("The Password your are provided "+password+" is valid");
        } else {
            System.out.println("The Password your are provided "+password+" is not valid");
        }
    }
}
```

8. Write a Java program to create a class called "Book" with instance variables title, author, and price. Implement a default constructor and two parameterized constructors:

a. One constructor takes the title and author as parameters.

b. The other constructor takes title, author, and price as parameters.

c. Print the values of the variables for each constructor.

```
public class Book {
    private String title;
    private String author;
    private float price;
    public Book() {
        title = "NA";
        author = "NA";
        price = 0.0f;
    }
    public Book(String title, String author) {
        super();
        this.title = title;
        this.author = author;
    }
    public Book(String title, String author, float price) {
        super();
        this.title = title;
        this.author = author;
        this.price = price;    }
```

```
    public static void main(String[] args) {
        Book b1 = new Book();
        Book b2 = new Book("Inkheart", "Cornelia Funke");
        Book b3 = new Book("Norwegian Woods", "Murakami", 270.00f);
    }
}
```

9. Write a Java program to create a class called Account with instance variables accountNumber and balance. Implement a parameterized constructor that initializes these variables with validation:

a. accountNumber should be non-null and non-empty.

b. balance should be non-negative.

c. Print an error message if the validation fails.

```
public class Account {
    private String accountNumber;
    private double balance;

    public Account(String accountNumber, double balance) {
        if (accountNumber != null && !accountNumber.isEmpty() &&
            balance >= 0) {
            this.accountNumber = accountNumber;
            this.balance = balance;
            System.out.println("Account created successfully!...");
            System.out.println("Your ACC NO is : " + this.accountNumber +
                               " with a balance of " + this.balance +
                               " rupees");
        } else {
            System.out.println("Failed to create an Account for the given acountNumber: " +
                               accountNumber);
        }
    }
    public static void main(String[] args) {
        Account acc1 = new Account("12345", 0.022);
        Account acc2 = new Account("54321", -0.226);        }           }
```

10. Write a Java program to create a class called Smartphone with private instance variables brand, model, and storageCapacity. Provide public getter and setter methods to access and modify these variables. Add a method called increaseStorage() that takes an integer value and increases the storageCapacity by that value.

```
public class Smartphone {
    private String brand;
    private String model;
    private int storageCapacity;

    public Smartphone(String brand, String model, int storageCapacity) {
        this.brand = brand;
        this.model = model;
        this.storageCapacity = storageCapacity;
    }

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public int getStorageCapacity() {
        return storageCapacity;
```

```
    }

    public void setStorageCapacity(int storageCapacity) {
        this.storageCapacity = storageCapacity;
    }

    public void increaseStorage(int amount) {
        this.storageCapacity += amount;
    }

    public static void main(String[] args) {
        Smartphone myPhone = new Smartphone("Apple", "iPhone 15", 256);

        System.out.println("Brand: " + myPhone.getBrand());
        System.out.println("Model: " + myPhone.getModel());
        System.out.println("Storage Capacity: " +
                        myPhone.getStorageCapacity() + " GB");

        myPhone.increaseStorage(128);
        System.out.println("Updated Storage Capacity: " +
                        myPhone.getStorageCapacity() + " GB");

        myPhone.setBrand("Samsung");
        myPhone.setModel("Galaxy S24");
        myPhone.setStorageCapacity(512);

        System.out.println("Brand: " + myPhone.getBrand());
        System.out.println("Model: " + myPhone.getModel());
        System.out.println("Storage Capacity: " +
                        myPhone.getStorageCapacity() + " GB");
    }
}
```