Basic Program

```
class Dog {
    String name;
    String breed;
    int age;

    void bark() {
        System.out.println("Woof! Woof!");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.name = "Buddy";
        myDog.breed = "Golden Retriever";
        myDog.age = 5;

        myDog.bark();
        System.out.println("My dog is named " + myDog.name);
    }
}
```

1.Encapsulation-It is defined as the wrapping up of data under a single unit. It is the mechanism that binds together the 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.
In encapsulation, the data in a class is hidden from other classes, which is similar to what **data-hiding** does. So, the terms “encapsulation” and “data-hiding” are used interchangeably.

```
class Account {
    private double balance;

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }

    public double getBalance() {
        return balance;
    }
}

public class Main {
    public static void main(String[] args) {
        Account myAccount = new Account();
        myAccount.deposit(1000);
        System.out.println("Account balance: " + myAccount.getBalance());
    }
}
```

2. Abstraction

- **Definition:** Hiding complex implementation details and showing only the necessary features of an object.
- **Abstract Classes and Interfaces:**
  - **Abstract Class:** Cannot be instantiated and may contain abstract methods that must be implemented by subclasses.
  - **Interface:** A reference type in Java, it is a collection of abstract methods.

```
abstract class Animal {
    abstract void sound();
}

class Dog extends Animal {
    void sound() {
        System.out.println("Woof! Woof!");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.sound();
    }
}
```

```
interface Animal {
    void sound();
}

class Dog implements Animal {
    public void sound() {
        System.out.println("Woof! Woof!");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.sound();
    }
}
```

3. Inheritance

- **Definition:** Mechanism where a new class inherits the properties and behaviors of an existing class.
- **Types of Inheritance:** Single, Multilevel, Hierarchical, (Java doesn't support multiple inheritance with classes but can be achieved with interfaces).

```
class Animal {
    String name;

    void eat() {
        System.out.println("Eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Woof! Woof!");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.name = "Buddy";
        myDog.eat();
        myDog.bark();
    }
}
```

4. Polymorphism

- **Definition:** The ability of an object to take many forms, allowing one interface to be used for a general class of actions.
- **Types:** Compile-time (Method Overloading), Runtime (Method Overriding).

Example (Method Overloading):

```
class MathOperations {
    int add(int a, int b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
}
public class Main {
    public static void main(String[] args) {
        MathOperations math = new MathOperations();
        System.out.println("Sum of two: " + math.add(10, 20));
        System.out.println("Sum of three: " + math.add(10, 20, 30));
    }
}
```

Example (Method Overriding):

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}
class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Woof! Woof!");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Dog();
        myAnimal.sound();
    }}
```
—----------------------- Other Oops Concepts—-------------------------------

Static Keyword

- Static Variables: Belong to the class rather than instances.
- Static Methods: Can be called without creating an instance of the class.
- Static Block: Code block that runs when the class is loaded.

```
class MathOperations {
    static int add(int a, int b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        int result = MathOperations.add(5, 10);
        System.out.println("Result: " + result);
    }
}
```

Final Keyword

- Final Variables: Cannot be changed after initialization.
- Final Methods: Cannot be overridden by subclasses.
- Final Classes: Cannot be subclassed.

```
class Constants {
    static final double PI = 3.14159;
}

public class Main {
    public static void main(String[] args) {
        System.out.println("Value of PI: " + Constants.PI);
    }
}
```

10. Exception Handling

- **Definition: Mechanism to handle runtime errors.**
- **Key Concepts: Try, Catch, Finally, Throw, Throws.**

```
public class Main {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero!");
        } finally {
            System.out.println("Execution complete.");
        }
    }
}
```